1995

Numerical investigation of Type II non-Newtonian de/anti-icing fluid effects on take-off performance for general aviation aircraft

Dennis James Cronin
Iowa State University

INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps. Each original is also photographed in one exposure and is included in reduced form at the back of the book.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

UMI

A Bell & Howell Information Company
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
313/761-4700 800/521-0600

Numerical investigation of Type II non-Newtonian de/anti-icing fluid effects on take-off performance for general aviation aircraft

by

Dennis James Cronin

A Dissertation Submitted to the

Graduate Faculty in Partial Fulfillment of the

Requirements for the Degree of

DOCTOR OF PHILOSOPHY

Department: Aerospace Engineering and Engineering Mechanics
Major: Engineering Mechanics

Approved:

Signature was redacted for privacy.

Signature was redacted for privacy.

In Charge of Major Work

Signature was redacted for privacy.

For the Major Department

Signature was redacted for privacy.

For the Graduate College

Members of the Committee:

Signature was redacted for privacy.

Signature was redacted for privacy.

Signature was redacted for privacy.

Iowa State University
Ames, Iowa

1995

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

Aircraft structural icing has been a problem that pilots have dealt with for many years. In-flight icing has long been known to cause problems with ice accruing mainly near the leading edge of aircraft under environmental conditions that include high water content and cold ambient temperatures. The adverse effects of accumulated in-flight ice are well known [2] [3] [4] [7] [34] [36]. If ice accumulates near the leading edge, it can form a "horned-shape" that effects the outer flowfield not to mention increasing the likelihood of separation as the modified flow passes around the accumulated ice, destroying the performance capabilities of the airfoil. The added weight of accumulated ice can be significant as well requiring higher temporary lift.

In-flight icing has its solutions, though. Pneumatic boots attached to the leading edge of lifting surfaces can cyclically inflate impulsively to expand and remove unwanted ice accumulation prior to anticipated aerodynamic problems [12]. On larger aircraft, excess bled air from the engines can be used to heat the leading edge to remove the ice. These methods, when used correctly, have avoided many potential accidents attributed to aircraft icing. Despite these efforts, in-flight icing has contributed to numerous accidents including recent events in Roselawn, Indiana with an ATR-42 in November 1994, and another incident in Raleigh, North Carolina involving a British Airways commuter plane in December 1994.

In-flight icing is a problem that will remain in the near future and its importance will grow. As excess bleed air from jet engines becomes smaller and smaller due to higher bypass ratios, the need for alternative methods of ice protection becomes greater. The formation of in-flight icing is very dependent on the initial conditions of the airfoil. If the wing is already covered with frost prior to flight, it is fair to assume that the lifting body is that much closer to accumulating ice and that much closer to disaster. As such, there are rules which state that pilots are required to check for ice prior to take-off and are, in fact, not allowed to take off under contaminated conditions. Currently, the sole responsibility for insuring clean surfaces lies with the pilot under these potential icing conditions (Federal Aviation Regulations (FAR) 135.227, 121.629(b) and 91.209 for large aircraft). Unfortunately, contrary to efforts by the Federal Aviation Administration (FAA) through pilot education, ground ice or frost accumulation is often overlooked by pilots in an effort to maintain schedules and minimize costs associated with deicing.

Ground icing is a separate problem from in-flight icing. It is estimated that 40 to 50 civil accidents per winter are attributed to icing, with half of those happening at take-off [25]. Loss of lift is considered to be the main problem. As an example, with only 0.5 mm of rime ice accumulation on the entire surface of a wing, the maximum lift coefficient fell 33% and the critical angle of attack fell from 13° to 7° under one typical hardwing configuration [24]. Even a small amount of frost is considered to be detrimental to aircraft performance. A frost layer of less than 1 mm has been shown to destroy up to 30% of lift [24].

The main method of preventing ground ice formation is through the use of freezing point depressant fluids applied directly to the bodies of aircraft. There are two types of fluids that are in use. They can be generally classified as either "deicing" or "anti-icing" fluids. The Type I fluid (as designated by the Society of Automotive Engineers (SAE)), a "deicing" fluid, is a mixture of glycol and water that is warmed to 60°C (140°F) and applied under pressure to clean the ice and snow off a wing or horizontal stabilizer. The Type I fluids are known to be linear or Newtonian meaning the shear stress developed within these fluids is linearly proportional to the amount of shear strain. The application procedures for this method are outlined for the operators [37]. Holdover time is the time after application of the de/anti-icing fluid to the aircraft under which it is considered safe to attempt a take-off. At the end of the holdover time, the fluids must be reapplied. Factors contributing to holdover times include outside ambient air temperature, temperature of fuel, and the various weather conditions e.g. frost, freezing fog, snow, freezing rain, and rain on cold soaked wing. After the application of the Type I fluid, the aircraft is considered to be free of any adverse ice or frost that might be present thus fulfilling the pilot's responsibility to have a clean wing under take-off conditions. Type I fluids are not known for their holdover times, but rather their ability to be applied easily and actually remove the fluids. The ease in application of these fluids comes partially from their relatively low viscosities when compared to the Type II fluids.

With holdover times at airports becoming increasingly longer, the current trend is toward de/anti-icers that have longer protection time in the event of poor weather conditions. Recently, a push has come from our European neighbors to use a fluid that is considered superior in ice protection. The new Type II fluids have been introduced in this country as a means of increasing the time a plane is protected from ground ice development. In contrast to the Type I fluids that are typically used in North America, the Type II fluid outperforms its counterpart by a wide margin . A typical Type I fluid may give 15 minutes of "protection" in steady snowfall, while the undiluted Type II fluid is valuable for about 45 minutes in falling snow conditions [19]. Type II fluid is also a water and glycol mixture, but includes a polymer

as a thickening agent. The fluid, which has the consistency of molasses, is not always heated before it is applied. It adheres to the airfoil rather than running off. If the wing already has ice or snow on it, this must be cleaned off first using the Type I fluid. During takeoff roll, ideally, the fluid blows off the airfoil to leave a clean surface.

Although the Type II fluids show promise for greater periods of ice protection, there are, of course, tradeoffs involved. The fluids that are present on the airfoil prior to takeoff generally shear off when the speed of the aircraft is great enough, leaving a clean surface for which the aerodynamic characteristics are well known. The problem comes when speeds obtained are such that the fluid does not completely shear from the airfoil. When this happens, essentially, the pilot is flying an untested airfoil about which the aerodynamic characteristics are generally not known.

The ideas about the effect of residual de/anti-icing fluid are summarized by Kenneth W. Hoefs, Boeing's director of airplane performance/airworthiness and chairman of a working group charged with studying the phenomenon:

> *The aircraft design working group noted that when a gel-like Type II anti-icing fluid is applied to an aircraft, not all of the fluid flows smoothly from the wings on takeoff....Residue generally results in measurable lift losses and drag increases during takeoff.*[26]

Concern is raised in the use of Type II fluids for general aviation because of the relatively slow speed of the aircraft. Here, the fluid may not completely shear from the wing giving a clean surface.

Another factor in the use of deicing fluids is the environmental impact these fluids might have as they run off the plane. One indication of this is restrictions on the amount of fluid that can be used freely without concern to fluid recovery. In Minneapolis, the airport is situated near the Minnesota River and it was found that excess deicing fluid that was being sent to the river in concentrated levels had potentially caused a lack of oxygen in the water required for sustained aquatic life under the frozen river. As a result, the amount of concentrated deicing fluid that can be released to the river is now restricted. A fluid recovery system is planned thus increasing the costs associated with deicing at this location [27].

Several factors now push the use of freezing point depressants to their limits. Both financial reasons and environmental concerns are bound to push the frontiers of the deicing fluids with consumers crying for greater holdover times. The trend has been to accept more viscous fluids which are greater in protection than the less viscous Type I fluids. As the

relative fluid viscosity increases, the aerodynamic performances are expected to suffer. For these reasons, if the use of Type 2 fluid is destined to increase in this country on smaller and smaller aircraft, a better understanding of the aerodynamic effects of residual fluid on airfoil surfaces is necessary.

2. PROBLEM BACKGROUND

2.1 Industry standards

At what point during the takeoff profile does the fluid shear off? What does this fluid do to the performance characteristics of the airfoil during this time? Are some airfoils more susceptible than others to performance degradation? What kinds of other parameters are important? Temperature effects? Thickness and uniformity of fluid application? Hard wing vs. slatted wings? These are all questions that need to be investigated to complete the picture in understanding the effects of residual fluid on airfoil performance.

There is a variety of fluids that are available on the open market. Killfrost from Arco, Octagon, Union Carbide and DOW FLIGHTGUARD are just several of many deicing fluids that meet the specifications set by SAE [38]. Each of these materials has different shear stress-rate of strain relationships and various abilities to protect against frost/snow/ice. Currently, the method by which de/anti-icing fluids are aerodynamically approved is through an experimental test which measures the displacement thickness accumulation at various points on a flat plate covered with a specified depth of fluid in a take-off simulation. The freestream velocity is accelerated linearly to a rotation speed and then allowed to settle at a constant value. If the displacement thickness remains small and under specifications, the fluid is approved for use. That is, the fluid is deemed aerodynamically safe.

2.2 Methods of exploration

2.2.1 Experimental

The best way to insure aerodynamic acceptance of de/anti-icing fluids is through the completion of actual flight testing. Apply the fluids in a manner that is representative of actual procedures, and make an effort to take off with a normal procedure. At the temperature and under environmental conditions of the test, either the plane will take-off or it won't. There are obvious flaws with this line of inquiry. Safety warrants that both aircraft and crew are not placed in jeopardy of a failed test. There have been several documented flight tests for a variety of deicing fluids [16] [40], but these were mainly for large aircraft where the fluids were almost assuredly not going to pose a safety danger. In these tests, the Boeing work was

completed on a 737-200ADV. Lift losses were measured at 5-12% depending on such factors as angle of attack and temperature (the deicing fluids would have different shear-rate of strain relationships at varying temperatures). The Fokker tests were completed on F-50 and F-100 aircraft with similar losses in lift as the Boeing tests; the F-50, a smaller aircraft, exhibited slightly more performance loss than the larger F-100.

The costs of all of these flight tests was great. Boeing and Fokker are two of the world's largest manufacturers of large aircraft. To flight-test all new fluids becomes cost-prohibitive for the fluid manufacturer. The aerodynamic acceptance test for deicing fluids was developed in parallel to the Boeing testing when the relationship between the boundary layer displacement thickness and aerodynamic characteristics of the a deicing fluid interacting with a airfoil was determined. This provides a cost-effective way for the fluid manufacturer to achieve certification.

There are some real limitations to this test, however. The rotation speed used in the SAE aerodynamic acceptance test is relatively large (65 ± 5 m/s) whereas the rotation speed of smaller aircraft is much less than this value. Take a Beech Baron for example- the take-off speed of the Baron, a four to six passenger single engine piston driven aircraft, is about 46 m/s. Since the deicing fluids have smaller relative viscosities at higher shear rates, it is safe to assume that during the acceleration phase of the take-off, more of the fluid is sheared off the airfoil at the highest speeds both from the standpoint that the magnitude of the shear forces is increasing and that the resistance to this force is decreasing. The last 24 m/s in this example would shear the greatest amount of fluid from the airfoil. This kind of test is to be expected as the SAE test for aerodynamic acceptance is based on work provided by Boeing for large transport category aircraft. Would it be fair to say that the fluid will not have significant aerodynamic penalty at lower rotation speeds? To this date, there have not been flight tests to answer this question and a corresponding new acceptance criteria developed for smaller aircraft. In the event that full-scale testing is not practical, aerodynamic engineers have often turned to the wind tunnel for answers.

In another study, perhaps the most complete to date, engineers at deHavalland in Canada made a series of detailed force measurements for a one-fifth scale model of a DHC-8 Series 100 in their two-dimensional wind tunnel [10]. Their measurements showed the same 10-12% lift loss as in the Boeing experiments. Increases in drag values from clean to contaminated surfaces were also shown in this test. The wide variety of influential parameters on the aerodynamic effect of the deicing fluids also raises the cost associated with the experimental work. As stated above, conditions such as air temperature, temperature of the

airfoil skin, water content in the air and a host of other environmental conditions make it a formidable task to say that "Yes, this deicing fluid is aerodynamically safe for use under all conditions."

There are many considerations when performing a wind-tunnel test. Namely, the first objective in a test, once the length scale has been chosen, is to make an effort to maintain not only geometric similarity, but kinematic similarity in order to insure dynamic similarity. If compressibility is not thought to be important (as it probably isn't in take-off performance analysis), then in order to insure kinematic similarity, the Reynolds numbers (based on whatever characteristic length one chooses) must be similar between model and prototype. In the tests conducted by the engineers at deHavalland, they are forthright in stating that there are difficulties in achieving this goal in a wind tunnel test of deicing fluid aerodynamic effects when the test is not full-scale. The non-linearity of the deicing fluid makes the chore difficult as the shearing characteristics of the deicing fluid change as a function of the shear rate. Even if the Reynolds number for the air is matched, the shear rates of the deicing fluid will not, in general, be the same between model and prototype. This is seen in the usual definition of Reynolds number. Although the velocity and characteristic length may be scaled correctly, the kinematic viscosity, which appears in the denominator of the Reynolds number, will not be the same. In a Newtonian fluid such as air or water, the kinematic viscosity is constant at a given temperature and there is no problem in maintaining kinematic similarity between model and prototype. What has been done is to complete a *distorted model* test. If all independent non-dimensional variables are similar between model and prototype, it is safe to assume that the dependent variable will be similar between model and prototype. In the distorted model, at least one of the independent non-dimensional parameters is not similar between the model and prototype and the conclusion is that we cannot say, without a doubt, that the independent non-dimensional parameter is equivalent between model and prototype. The options at this point are to 1) ignore the influence of the non-similar variable or 2) try and account for the effect of the non-similar variable in an analytical manner.

There are examples of distorted models providing useful information (modeling of rivers where the horizontal length scale is different than the vertical length scale as to avoid surface tension effects in the model is one such example), but in the end, there is always an element of doubt placed on the results.

Another limitation of wind-tunnel testing is the inability to provided large enough Reynolds numbers even if the model were not distorted. Take the Baron again; at standard atmospheric values for the air, the take-off Reynolds numbers based on an average chord

length (c = 1.54 m) and the kinematic viscosity of air, are about $4.8(10^6)$. Here at Iowa State in the two-dimensional tunnel, the maximum Reynolds numbers we can achieve are about $2.5(10^6)$ without having to concern ourselves with tunnel blockage effects at higher angles of attack. We would have difficulty in providing the required Reynolds numbers. To model larger aircraft, both the take-off speed and the chord length increase so as to make the chore of matching take-off Reynolds numbers even more difficult.

2.2.2 Numerical

Fortunately, in recent years, the growth in the abilities of newer computers has made methods of numerical approximation practical. Numerical approximation of a physical phenomena has several advantages over the actual experimental work, particularly when practical experimental work becomes cost-prohibitive.

The equations of motion which describe a physical phenomena can be approximated by a series of numerical calculations that may include boundary element methods, finite element methods or finite difference methods. As a numerical grid encompasses the spatial and temporal coordinates, it is hoped that with finer resolution of the meshing, the numerical solutions will give more accurate approximations to the differential equations. There are those that would have you believe that the numerical method approach is fully capable of arriving at correct solutions to complex flow phenomena, but the correct modeling of the physical situation is critical in determining the best solutions. One may start with a full-blown time-dependent Navier-Stokes set of equations, but the computational resources become increasingly taxing at that point. Modeling such complex flow developments as turbulence often require the use of empirical data and regardless of how good you believe the modeling to be, there is no replacement for actual full-scale experimental work. Nature often has a way of hiding aspects of the flow that can hardly appear in any modeling. Numerical methods can be used for relatively quick design iterations, however. Once a model is built, the modifications that are necessary to change a design are minimal in comparison to a new prototype that needs to be built. As such, numerical methods are particularly good in the conceptual and preliminary design phases of a project.

Such is the situation in the understanding of the effect of the deicing fluid on the aerodynamic performance of a lifting surface. In this case, the path for practical experimental work is not clear. Either a distorted model is used or the costs of a full-scale test are accrued. Even so, each full-scale test would be for only one specific set of parameters: section

geometry, ambient temperature, shear-rate of strain characteristics of the fluid, and even take-off procedure.

2.3 Problem statement

With the present limitations of a practical experimental understanding of the effects of deicing fluid on the aerodynamic properties of lifting surfaces, the premise and need for numerical approximations of the phenomena is in place. What this study will attempt to do is through an "engineering" approach, look for trends in the effect of deicing fluid on the aerodynamic performance of lifting surfaces under typical take-off procedures. It is expected that parameters such as section geometry, deicing fluid dynamic viscosity, initial deicing fluid depth, and take-off procedure in the form of initial angle of attack, acceleration rate of the freestream velocity, and pitch rate upon rotation will lead to a measure of maximum lift coefficient. Specifically, in the development of the lift coefficient for a particular geometry, there is some maximum value that will occur at a critical angle of attack that, should the angle of attack be pushed beyond this, the flow will separate. The conditions for a flow separation are defined later, but at that point, it is assumed, the airfoil is no longer producing maximum lift. There is also an increased amount of drag due to pressure drag, but in this study, the maximum lift coefficient will be the primary parameter of interest.

In performing an "engineering" analysis of the phenomena, there will be several approximations made that should be taken into account when viewing the results:

1. First of all, a two-dimensional approach shall be attempted. The airfoil geometry will be assumed to be infinitely long into the page, and the usual two-dimensional assumptions will be made. As an attempt at better understanding the effects of the fluid on the aerodynamic performance during take-off, a three dimensional analysis is recommended for further work.
2. Outer flow solutions will be obtained via a modified PANEL method that will involve deviations from the conventional steady-state PANEL method in that time-dependency of pressure coefficients is established and the "no penetration" condition is modified to account for trailing vortices and displacement thicknesses.
3. Gas-dynamic boundary layers are solved with attention to modifying the stagnation point solution and paying special attention to transition to turbulence criteria. Also, within the gas-dynamic boundary layer, the equations of motion are simplified to the boundary layer equations which will be justified later.

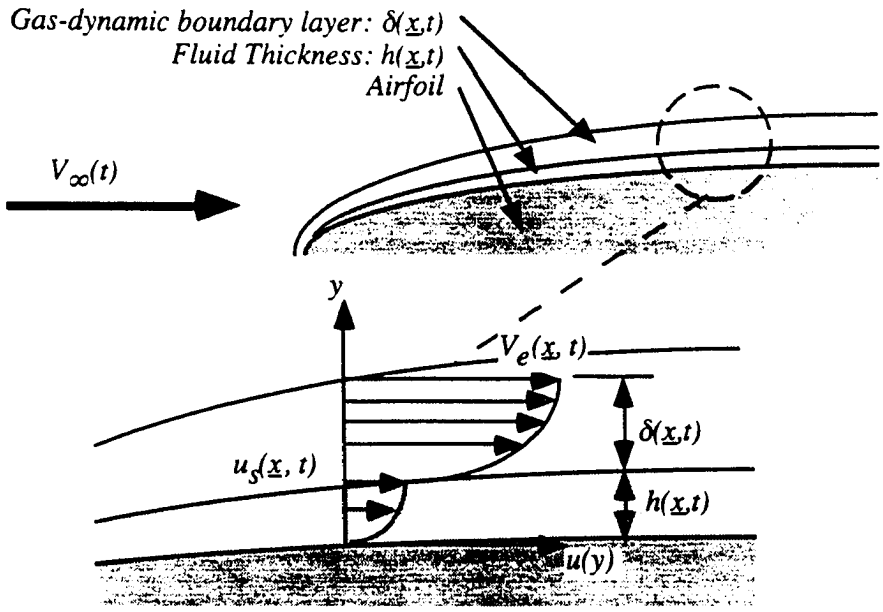


Figure 2.1: Introduction to the double boundary layer problem

4. Within the part of the program that describes how the deicing fluid reacts to the forces due to shear stress and pressure gradient, the fluid will be assumed to have minimal inertia. The dynamic viscosity of the deicing fluid is modeled as Newtonian or linear, with the non-linear behavior recommended as available for further study.

With the need for a computational approach justified and some of the simplifying assumptions listed, we are ready to formulate the problem, keeping in mind that the ultimate measure of performance will be a maximum section lift coefficient obtainable by the airfoil. Recall that it is the general aviation aircraft that has relatively low rotation speeds that we are most interested in. If the speed of the aircraft is not great enough, the fluid will not shear from the airfoil and the aerodynamic performance of the airfoil is expected to suffer.

2.4 Formulation of the problem: the double boundary layer

The implicit problem being solved in this research is that of a two level boundary layer, one that consists of the de/anti-icing fluid which varies as a function of position and time and a second that consists of the gas-dynamics. Figure 2.1 shows an appropriate delineation along with the parameters of fluid thickness, h, and the gas-dynamic boundary layer, δ.

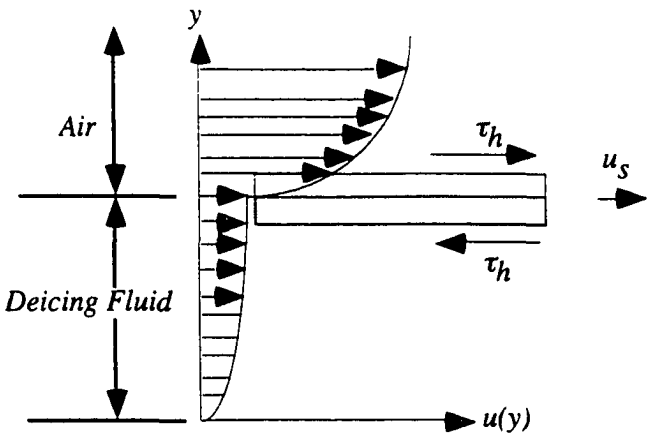


Figure 2.2: Matching conditions at interface of boundary layers

There have been several studies into the two-fluid interaction problem [9] [17] [21] [42] [43]. Most have had limiting assumptions in that either the driving flow was steady, the shear stress distribution that drives the motion of the thin film is known prior to the study (as in a laminar boundary layer for steady flow), or that the geometry of the problem was fixed such that there was not a pressure gradient in the thin-film.

Another assumption in these studies, one that seems perfectly valid in the engineering approach that we have chosen, is that the ratio of viscosities between the driving fluid (air in this case) and the thin-layer fluid was very small. What this last assumption does for us is to allow us to neglect contributions from inertia within the thin-fluid layer. This assumption will be used with comment in further analysis.

2.5 Matching conditions of kinematic and dynamic continuity at the interface

Implicit to the interaction of the dual boundary layer is matching conditions at the interface of the gas-dynamic boundary layer and the deicing fluid. Figure 2.2 shows the component of velocity parallel to the airfoil. Assume that there is one particle of deicing fluid that is in contact with one particle of air. For these particles to remain in contact, there are two conditions that must be matched at the interface. One is for kinematic matching, the other for dynamic matching. The kinematic matching condition is such that at the interface, both the deicing fluid and the air must have the same horizontal component of velocity, u_s, as shown in the figure. The second matched condition is the dynamic matching condition, namely that of an

equivalent shear stress at the interface, shown as τ_h in the figure. The usual no-slip condition of zero velocity is modified in the gas-dynamic boundary layer. It is believed that this may be a critical component of the research contained in this study. How does the modification of the no-slip condition effect the transition point (point at which the flow changes from laminar to turbulent) of the airfoil? What kind of effects can be expected from the presence of the deicing fluid in terms of accounting for a displacement thickness associated with the growth of the boundary layer?

There are now three pieces to the puzzle that must be solved. First, there is an outer flow that is governed by a potential flow approach (one that does not account for the effects of viscosity), there is the solution of the gas-dynamic boundary layer growth and finally, an attempt must be made to look at the deicing fluid. Once these individual components have been solved, the process of integrating the total package will give an initial picture of the performance of the airfoil and the modified performance due to the presence of the deicing fluid. The integration of the various components will make use of the matching conditions outlined above and an effort will be made to account for the effect of displacement thickness.

3. METHODS OF ANALYSIS: THE OUTER FLOW

3.1 Potential flow

Potential flow problems are common in the study of aerodynamics. The underlying assumption in potential flow is that there are no effects from the viscosity of the fluid. There are several numerical methods that can be used to calculate the potential flow about a two-dimensional airfoil, for example. Finite element methods are popular in that there exists much experience in this relatively old field. The difficulty in using finite elements is that there is a choice that must be made as to where the point of influence of the body exists. In a finite element method approach, the modeled equation requires boundary conditions such that "far away" from the airfoil, the effect of the body ceases and the flow approaches that of the freestream. This decision brings an element of doubt into the analysis that is not present with another potential method, the Boundary Element method, sometimes called the Boundary Integral Equation method.

In the Boundary Element method, there is still a need to discretize the domain of the body in the freestream, but in this case, the boundary conditions are such that 1) a no penetration condition is satisfied on the surface of the body and 2) the "far away" condition is automatically satisfied in the mathematical formulation of the problem. No estimates are necessary as to the point where the influence of the body ends. The most common implementation of using the Boundary Element method is summarized in the PANEL method. Hess and Smith [15] are generally credited with this formulation, but others have successfully applied the method for calculating potential flow around airfoils in two dimensions and even other fully discretized bodies in three dimensions such as aircraft and road vehicles. The realization of the implementation of the PANEL method is that the pressure effects of an oncoming stream on a body can be modeled fairly accurately and simple calculations such as lift and drag coefficients are close to values measured in the wind tunnel or in-flight. It is not until viscous effects are accounted for (in other methods) that the complete picture of performance can be accurately predicted, but the potential flow predictions give a good approximation with respect to tangential or edge velocities and even pressure distributions which constitute a majority of the estimated lift on an airfoil.

3.2 The PANEL method for steady flow

We will, in a first attempt, look at the development of the PANEL method for steady potential flow and eventually look at the modifications that are necessary to account for the time-dependence encountered in a take-off simulation as well as how to interpret the results with the time-dependence in mind.

The first step in applying the PANEL method is to determine the equation that is to be solved. Secondly, once the appropriate equation is defined and implemented, a condition must be placed on the flow as to provide a unique solution. Finally, a relationship between the primitive variables of velocity and pressure must be found.

Under the assumptions of two-dimensional inviscid, incompressible, irrotational flow, if a the velocity vector, $\underline{V}$, is defined as the gradient of the velocity potential, ϕ (a scalar),

$$\underline{V} = \nabla\phi = \frac{\partial\phi}{\partial x}\hat{i} + \frac{\partial\phi}{\partial y}\hat{j} = u\hat{i} + v\hat{j} \tag{3.1}$$

where $\hat{i}$ and $\hat{j}$ are unit vectors parallel to the fixed x and y coordinate system and u and v are components of the velocity vector, the general conservation of mass equation, with ρ being the fluid density,

$$\frac{D\rho}{Dt} + \rho(\nabla \cdot \underline{V}) = 0 \tag{3.2}$$

becomes

$$\begin{aligned} (\nabla \cdot \underline{V}) &= (\nabla \cdot \nabla\phi) \\ &= \nabla^2\phi \\ 0 &= \frac{\partial^2\phi}{\partial x^2} + \frac{\partial^2\phi}{\partial y^2} \end{aligned} \tag{3.3}$$

Thus we have one linear partial differential equation for a scalar, ϕ, namely Equation (3.3). Equation (3.3) is of second order and requires two boundary conditions for a unique solution. One of the boundary conditions is known to be the "no penetration" condition. This says that no flow is allowed to penetrate a body subject to the external flow. The second boundary condition is that "far away" from the body, the influence of the body must be negligible. As mentioned in Section 3.1, other numerical methods can have difficulty with this portion of the problem, but in the Boundary Element method, use is made of Green's Theorem

which, under special conditions, reduces a problem in two dimensions to one of only one coordinate, namely that of the surface of the airfoil.

Analytically, let there be a velocity potential, ϕ, that satisfies Laplace's Equation, for a two dimensional flow field such that

$$\phi = \phi_\infty + \phi_S + \phi_V \tag{3.4}$$

where

$$\phi_\infty = V_\infty(x\cos\alpha + y\sin\alpha) \tag{3.5}$$

represents the velocity potential due to the freestream velocity, V_∞, at angle of attack, α,

$$\phi_S = \oint_s \frac{q(s)}{2\pi}\ln(r)ds \tag{3.6}$$

where ϕ_S is the velocity potential due to a line source/sink distribution around the airfoil, and

$$\phi_V = -\oint_s \frac{\gamma(s)}{2\pi}\theta\, ds \tag{3.7}$$

where ϕ_V is a velocity potential due to a line vortex distribution around the airfoil surface. Here, in two dimensions, $q(s)$ is a source per unit length and $\gamma(s)$ is a circulation per unit length distributed around the surface of the airfoil. The coordinates r and θ are locally defined around the body so that the velocity potential is a function of the position in the flowfield (x, y) as shown in Figure 3.1. Note that the integration takes place along the surface in a clockwise manner. The objective is to solve Equation (3.3) by choosing $q(s)$ and $\gamma(s)$ such that the no penetration condition is met.

Recall that Equation (3.3), while being second order, is also what is called linear. In the general case, there are really an infinite number of solutions that are merely algebraic ratios of each other. Once a solution is found, then within an arbitrary coefficient, there exists a similar solution. The individual components of Equation (3.4) are general solutions to the Laplace Equation. Also, a feature of the linear problem is that once a second independent (not a solution that differs only by a leading coefficient) is found, it can be added to the first independent solution to form a third, just as appropriate solution. This is the idea of superposition that is carried out in Equation (3.4). It is not until a purely physical limitation is applied that a unique solution can be determined.

This physical limitation is known as the Kutta Condition.

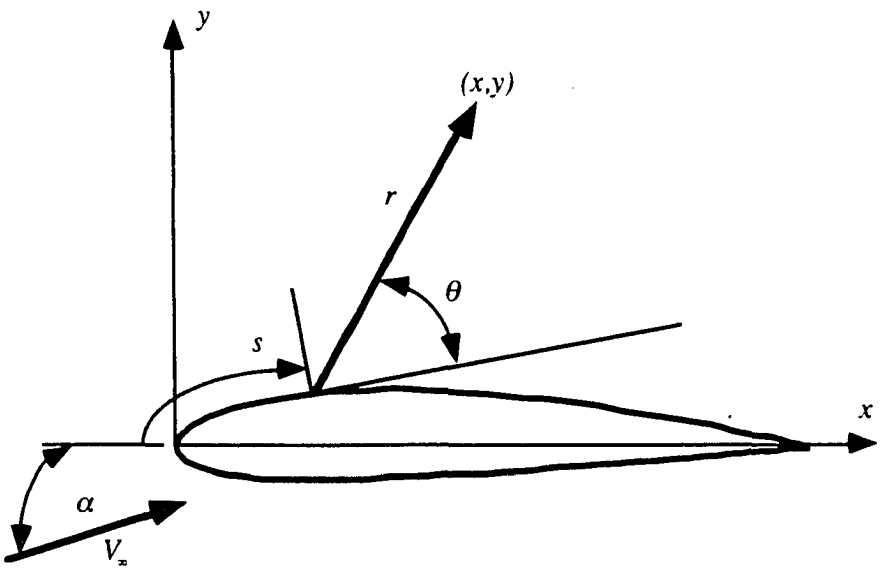


Figure 3.1: Nomenclature for surface singularity distributions

> ***Kutta Condition:*** *The flow leaves the trailing edge of a sharp-edged airfoil smoothly. That is, the velocity is finite there.*[29]

and the following corollary of the Kutta Condition, which is often more useful:

> *Near the trailing edge, the flow speeds on the upper and lower surfaces of the airfoil are equal at equal distances from the trailing edge.*

We will make use of this condition later in the numerical implementation of the PANEL method.

Finally, in order to interpret the meaning of the solution to Equation (3.3), we need to relate the primitive variables of pressure and velocity. Beginning with the two dimensional Euler equations for an incompressible, inviscid flow with no circulation, the momentum equation appears as

$$\rho \frac{D\underline{V}}{Dt} + \nabla p = 0 \tag{3.8}$$

where p is the static pressure. There are two component equations of Equation (3.8) when the flow is further assumed to be steady and two-dimensional, namely

$$\rho \underline{V} \cdot \nabla u + \frac{\partial p}{\partial x} = 0 \tag{3.9}$$

$$\rho \underline{V} \cdot \nabla v + \frac{\partial p}{\partial y} = 0 \tag{3.10}$$

Combining Equation (3.1) and Equation (3.9) we arrive at

$$\begin{aligned} -\frac{\partial p}{\partial x} &= \rho \nabla \phi \cdot \nabla \frac{\partial \phi}{\partial x} \\ &= \rho \nabla \phi \cdot \frac{\partial (\nabla \phi)}{\partial x} \\ &= \frac{\rho}{2} \frac{\partial}{\partial x} (\nabla \phi \cdot \nabla \phi) \\ &= \frac{\rho}{2} \frac{\partial}{\partial x} (V^2) \\ 0 &= \frac{\partial}{\partial x} \left(p + \frac{\rho}{2} V^2 \right) \end{aligned} \tag{3.11}$$

Similarly, we use Equation (3.1) and (3.10) to show that

$$\frac{\partial}{\partial y} \left(p + \frac{\rho}{2} V^2 \right) = 0 \tag{3.12}$$

If this value, $p + \frac{\rho}{2} V^2$, is independent of both x and y, then it is constant throughout the flowfield. Equations (3.11) and (3.12) combine to form the well-known Bernoulli Equation for steady, invicid, incompressible flow along a streamline. Once the velocity potential is found from the solution of Laplace's Equation, the velocity is determined by Equation (3.1) and the pressure is found from Equation (3.11) or (3.12).

3.3 Numerical discretization

With the goal of solving Laplace's Equation in two dimensions, we look to numerically approximate this equation (Equation (3.3)) at a set of discrete points. So, the first step is to discretize the body of interest. Referring to Figure 3.2, there are a series of panels that approximate the geometry of the airfoil. The scheme used in numbering the panels is such that each panel is made up of two nodes, the nodes beginning at number "1" at the trailing edge on

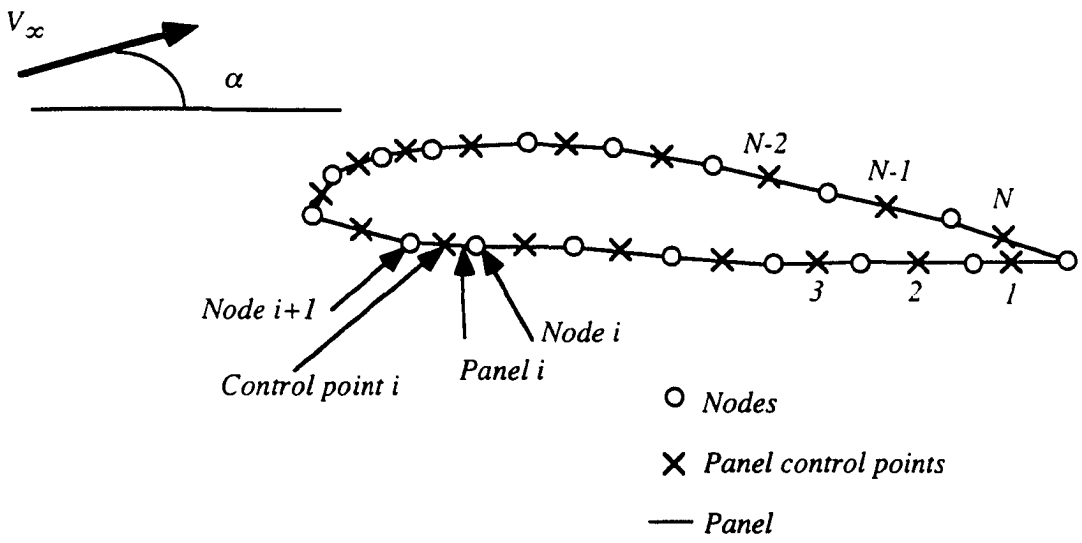


Figure 3.2: Numbering scheme for panels, nodes and panel control points

the bottom of the airfoil and increasing sequentially in a clockwise fashion until node "*N*+1" coincides with the placement of node "1" at the trailing edge. Between node "1" and node "2" exists the midpoint of the first panel. This continues until there are "*N*" panels that describe the geometry.

Figure 3.3 is indicative of the way chosen in this study to discretize the airfoil geometry. Shown in the figure is a NACA 2412 airfoil that is approximated by a series of panels with nodes at the end of each panel and a midpoint that will be used as a control point for enforcement of the boundary conditions. In Figure 3.3, the small "X" values are the panel midpoints placed at a value of $\bar{y}/c$, being a function of $\bar{x}/c$ for a particular airfoil where c is the chord length. The discretization is done such that the distance from the leading edge is given non-dimensionally in terms of the chord length and that thicknesses of the particular airfoil are also given non-dimensionally. This theme is consistent throughout the analysis. For example, the geometry is given in terms of "length/unit length". Similarly, velocities should be thought of as units length per unit time. Keeping this in mind will help in transferring the final results to a prototype airfoil. All lengths within this study should be thought of as "length relative to the chord length".

In this study, the panels are situated such that they are clustered near the leading and trailing edges of the two-dimensional lifting surface. This is because the gradients of the

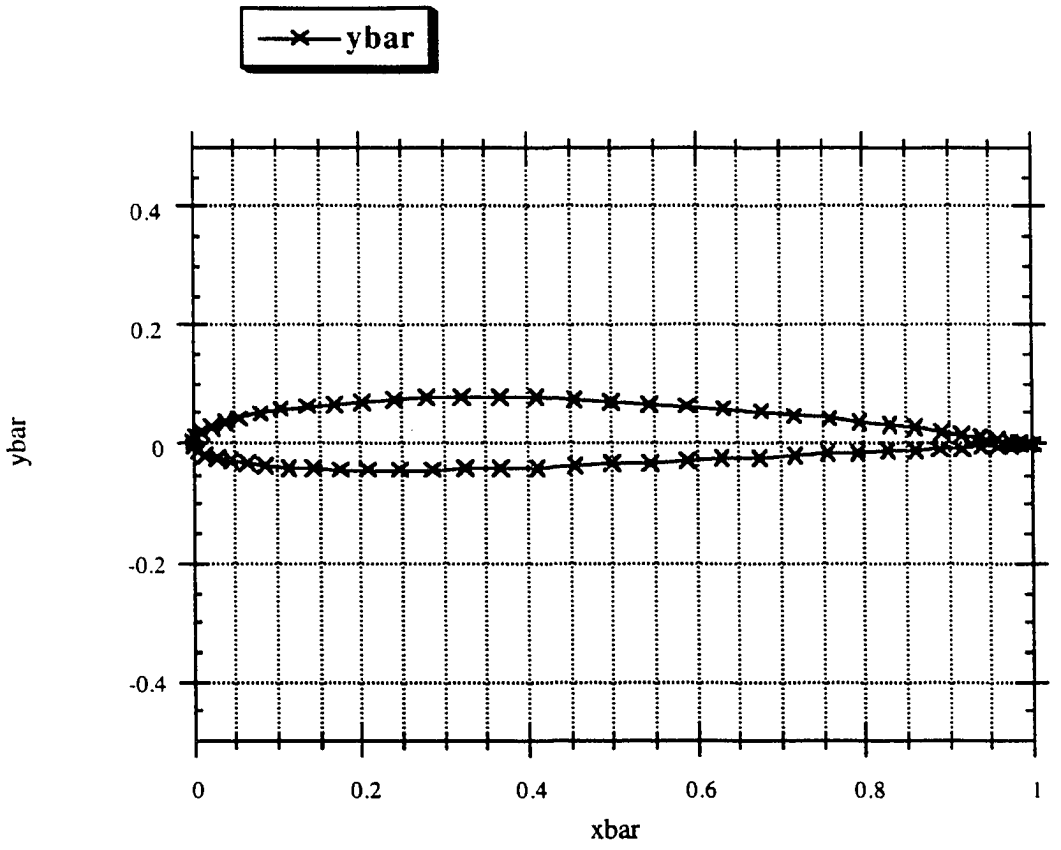


Figure 3.3: A typical airfoil discretization for potential flow analysis

tangential velocity (sometimes called edge velocities- V_e) are expected to be greatest in these regions. To be complete, the points are distributed using a cosine variation. With $fract = (i-1)/(nl)$ where nl is the number of nodes on the lower surface, $x(i) = \frac{1}{2}(1+\cos(-fract\,\pi))$. On the upper surface, if nu is the number of nodes on the upper surface, $fract = (i-1-nl)/(nu)$ and $x(i) = \frac{1}{2}(1-\cos(-fract\,\pi))$.

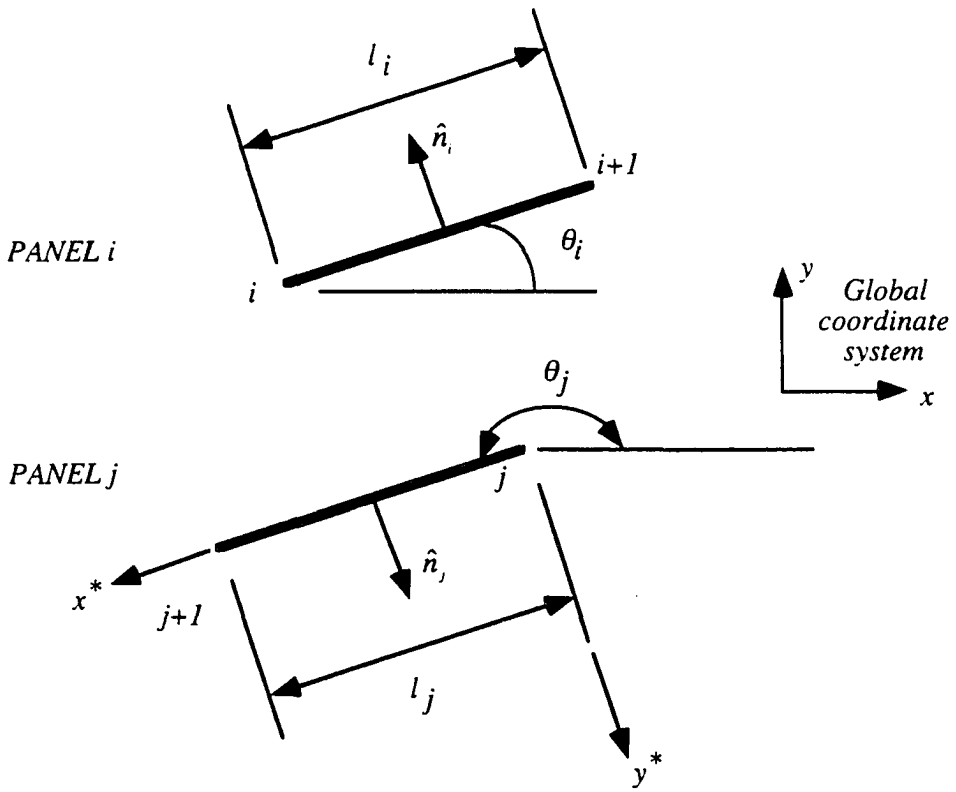


Figure 3.4: Panel geometry definitions

Figure 3.4 exhibits several important features of the paneling geometry. Each panel has a length and orientation. The angle θ_i is defined positive in the counter-clockwise sense relative to the global x axis and the length, l_i, for example, is the distance between node i and node $i+1$. This is shown in both panel i and j in Figure 3.4. Imagine that panel i is on top of the airfoil and that panel j is on the bottom noticing that the unit vector, $\hat{n}_i$, points outward from the body at all panels. Another item called out in Figure 3.4 is a set of local coordinates that is oriented with each panel. These are labeled as x^* and y^* in Figure 3.4. This reference frame will be used in the development of the equations to be solved.

3.4 Method of solution and numerical implementation of the PANEL method

Assume that in the two dimensional formulation of the problem there exists two panels, i and j (See Figure 3.5). Let panel i "feel" some sort of velocity influence from a fluid source, now distributed along panel j.

For N panels, the velocity potential that is analytically shown in Equation (3.4) can be approximated by

$$\phi = V_\infty(x\cos\alpha + y\sin\alpha) + \sum_{j=1}^{N} \int_{panel\ j} \left[\left(\frac{q(s)}{2\pi}\ln(r) - \frac{\gamma}{2\pi}\theta\right)ds\right] \tag{3.13}$$

Implicit in this formulation is that the source strength is allowed to vary from panel to panel and that since the Kutta Condition needs to be satisfied at only one point (the trailing edge), there is a single value for γ which is constant throughout the flow. Following Hess and Smith, we allow the source strength to vary from panel to panel but to be constant along each panel: $q(s) = q_i$ on panel i, $i = 1,\ldots, N$. The parameters to be determined are the source strengths, q_i and the vortex strength per unit length, γ. We apply the no penetration condition at the N control points situated at the midpoint of each panel and the Kutta Condition at the trailing edge to solve for the N sources and the γ value.

We need to formulate these conditions into a linear set of equations to be solved for the N+1 variables. Let

$$\begin{aligned} u_i &= u(\overline{x}_i, \overline{y}_i) \\ v_i &= v(\overline{x}_i, \overline{y}_i) \end{aligned} \tag{3.14}$$

be the (global) horizontal and vertical components of velocity at the midpoint of panel i. The no penetration condition, sometimes called the flow tangency condition in the steady analysis, is written as

$$0 = u_i\sin\theta_i + v_i\cos\theta \text{ for } i = 1,\ldots N \tag{3.15}$$

while the Kutta Condition is written as

$$u_1\cos\theta_1 + v_1\sin\theta_1 = -u_N\cos\theta_N + v_N\sin\theta_N \tag{3.16}$$

The question becomes how do we write the velocity components? Recall that the velocities induced by the sources and vortices are proportional to the strength of the source or vortex on that panel, so we can write

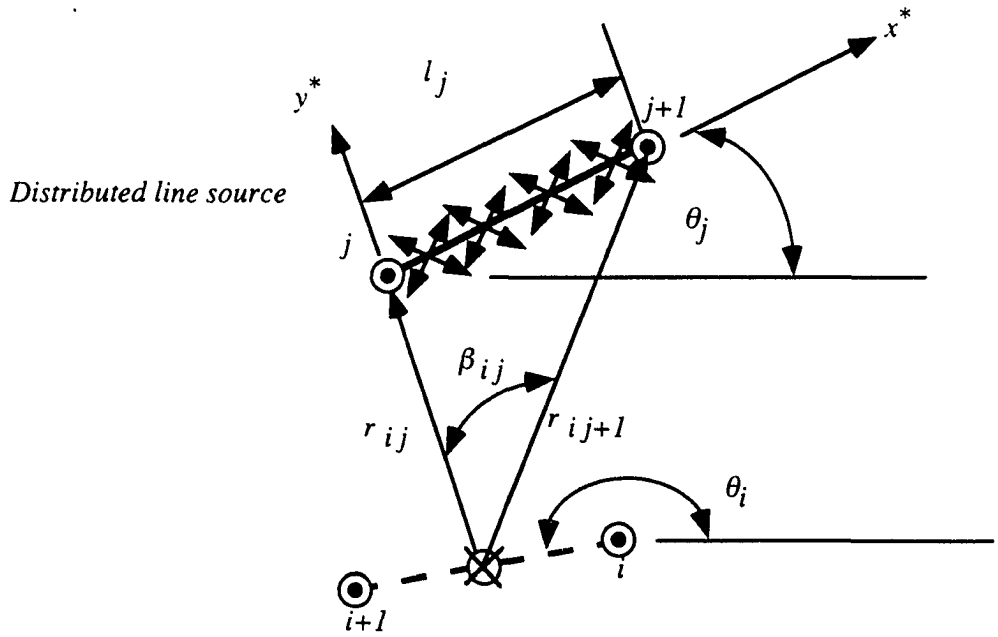


Figure 3.5: Coefficient setup nomenclature

$$u_i = V_\infty \cos\alpha + \sum_{j=1}^{N} q_j u_{sij} + \gamma \sum_{j=1}^{N} u_{vij} \tag{3.17}$$

$$v_i = V_\infty \sin\alpha + \sum_{j=1}^{N} q_j v_{sij} + \gamma \sum_{j=1}^{N} v_{vij}$$

where u_{sij}, for example, is the x component of the velocity at the midpoint of the ith panel due to a unit-strength source distribution on the jth panel. If the local coordinates are used at panel j as in Figure 3.5, the global velocities can be written as

$$u = u^* \cos\theta_j - v^* \sin\theta_j \tag{3.18}$$

$$v = u^* \sin\theta_j + v^* \cos\theta_j$$

once we find the "local" components of velocity, (u^*, v^*). Now, we look at the "local" velocities induced at node i due to a distributed unit source along panel j.

It can be shown that

$$u_{sij}^{\bullet} = \frac{-1}{2\pi}\ln\left(\frac{r_{ij+1}}{r_{ij}}\right) \qquad (3.19)$$

$$v_{sij}^{\bullet} = \frac{\beta_{ij}}{2\pi}$$

Notice that these terms depend only on the geometry. r_{ij+1}, r_{ij}, and β_{ij} are as shown in Figure 3.5. Similarly, if there is a distributed vortex along panel j of unit strength,

$$u_{vij}^{\bullet} = \frac{\beta_{ij}}{2\pi} \qquad (3.20)$$

$$v_{vij}^{\bullet} = \frac{1}{2\pi}\ln\left(\frac{r_{ij+1}}{r_{ij}}\right)$$

One point of interest comes when $i = j$. What is the included angle β_{ij}? Taking the path of integration to be such that the normal vector points outward and the integration is done in a clockwise direction, $\beta_{ij} = \pi$ rather than $-\pi$.

The flow tangency conditions (Equation (3.15))may now be put in the form

$$\sum_{j=1}^{N} A_{ij}q_j + A_{iN+1}\gamma = b_i \text{ for } i = 1,\ldots, N \qquad (3.21)$$

with

$$A_{ij} = \frac{1}{2\pi}\left[\sin(\theta_i - \theta_j)\ln\left(\frac{r_{ij+1}}{r_{ij}}\right) + \cos(\theta_i - \theta_j)\beta_{ij}\right] \qquad (3.22)$$

$$A_{iN+1} = \frac{1}{2\pi}\sum_{j=1}^{N}\left[\cos(\theta_i - \theta_j)\ln\left(\frac{r_{ij+1}}{r_{ij}}\right) - \sin(\theta_i - \theta_j)\beta_{ij}\right] \qquad (3.23)$$

and

$$b_i = V_\infty \sin(\theta_i - \alpha) \qquad (3.24)$$

The Kutta Condition (Equation (3.16)) can be put in the form:

$$\sum_{j=1}^{N} A_{N+1,j}q_j + A_{N+1,N+1}\gamma = b_{N+1} \qquad (3.25)$$

where

$$A_{N+1,j} = \frac{1}{2\pi}\sum_{k=1..N}\left[\sin(\theta_k - \theta_j)\beta_{kj} - \cos(\theta_k - \theta_j)\ln\left(\frac{r_{kj+1}}{r_{kj}}\right)\right] \tag{3.26}$$

$$A_{N+1,N+1} = \frac{1}{2\pi}\sum_{k=1..N}\sum_{j=1}^{N}\left[\sin(\theta_k - \theta_j)\ln\left(\frac{r_{kj+1}}{r_{kj}}\right) + \cos(\theta_k - \theta_j)\beta_{kj}\right] \tag{3.27}$$

and

$$b_{N+1} = -V_\infty\cos(\theta_1 - \alpha) - V_\infty\cos(\theta_N - \alpha) \tag{3.28}$$

This gives N+1 equations for the N+1 unknowns q_i (i = 1, ..., N)and γ. This can be solved with any linear equation solver. The coefficient matrix is dense, so it is of no use to use a method which can take advantage of the sparseness of the matrix. Using Gaussian decomposition, the unknowns are determined and the tangential, or edge, velocities can be found from

$$V_{e,i} = V_\infty\cos(\theta_i - \alpha) + \sum_{j=1}^{N}\frac{q_j}{2\pi}\left[\sin(\theta_i - \theta_j)\beta_{ij} - \cos(\theta_i - \theta_j)\ln\left(\frac{r_{ij+1}}{r_{ij}}\right)\right] \tag{3.29}$$

$$+\frac{\gamma}{2\pi}\sum_{j=1}^{N}\left[\sin(\theta_i - \theta_j)\ln\left(\frac{r_{ij+1}}{r_{ij}}\right)\beta_{ij} + \cos(\theta_i - \theta_j)\beta_{ij}\right]$$

In the steady flow problem, the pressure coefficient at the midpoint of panel i can now be calculated

$$C_p(\overline{x}_i, \overline{y}_i) = \frac{p - p_\infty}{\frac{1}{2}\rho V_\infty^2} = 1 - \left(\frac{V_{e,i}}{V_\infty}\right)^2 \tag{3.30}$$

Integration of the pressure coefficients around the surface of the airfoil gives the section lift and drag coefficients (c_l and c_d) which are the non-dimensional lift (component force perpendicular to the freestream) and drag (component force parallel to the freestream) values as in Equations (3.31) and (3.32) where L' is the lift per unit width and D' is the drag per unit width.

$$c_l = \frac{L'}{\frac{1}{2}\rho V_\infty^2 c} \tag{3.31}$$

$$c_d = \frac{D'}{\frac{1}{2}\rho V_\infty^2 c} \tag{3.32}$$

$$c_{m,l.e.} = \frac{M'}{\frac{1}{2}\rho V_\infty^2 c^2} \tag{3.33}$$

Equation (3.33) shows a non-dimensional moment coefficient, $c_{m,l.e.}$. In this case, the moment is taken about the leading edge, but there are other points that are convenient such as the quarter chord.* In Equation (3.33), M' is a moment about the leading edge per unit width. These non-dimensional parameters are modified slightly by the inclusion of the resultants due to the shear stress distribution that results on the airfoil when viscosity is accounted for.

3.5 Modification to the PANEL method due to time-dependence

There exist several modifications that are necessary when talking about the accelerating freestream and rotation maneuvers associated with a take-off simulation. The modifications come in how to interpret the pressure coefficients and how to account for a time dependence of the total flowfield circulation in both the manner of "vortex shedding" and in the modification to the PANEL method.

Begin, again, with the time-dependent Euler equation:

$$\rho \frac{D\underline{V}}{Dt} + \nabla p = 0 \tag{3.8}$$

and retain the time-dependent terms. If the components of velocity are defined as in Equation (3.1)

$$\underline{V} = \nabla\phi = \frac{\partial\phi}{\partial x}\hat{i} + \frac{\partial\phi}{\partial y}\hat{j} = u\hat{i} + v\hat{j} \tag{3.1}$$

then the component equations become

---

* The author would like to recognize Dr. Jack Moran formerly at the University of Minnesota who not only served as an inspiration to this author to study computational fluid dynamics, but also wrote the text in which the procedure for the steady flow analysis used in this study is clearly outlined and from which this author heavily borrows.

$$\rho\frac{\partial u}{\partial t}+\rho\underline{V}\cdot\nabla u+\frac{\partial p}{\partial x}=0 \tag{3.34}$$

$$\rho\frac{\partial v}{\partial t}+\rho\underline{V}\cdot\nabla v+\frac{\partial p}{\partial y}=0 \tag{3.35}$$

Working with Equation (3.34) only, with the definitions used in Equation (3.1), we get

$$\rho\frac{\partial}{\partial t}\left(\frac{\partial\phi}{\partial x}\right)+\rho\left[(\nabla\phi)\cdot\nabla\left(\frac{\partial\phi}{\partial x}\right)\right]+\frac{\partial p}{\partial x}=0 \tag{3.36}$$

Since the order of differentiation is not important, this can be written as

$$\rho\frac{\partial}{\partial x}\left(\frac{\partial\phi}{\partial t}\right)+\frac{\rho}{2}\frac{\partial}{\partial x}\left[(\nabla\phi)\cdot(\nabla\phi)\right]+\frac{\partial p}{\partial x}=0 \tag{3.37}$$

$$\frac{\partial}{\partial x}\left[\rho\left(\frac{\partial\phi}{\partial t}\right)+\frac{\rho}{2}\left[(\nabla\phi)\cdot(\nabla\phi)\right]+p\right]=0$$

$$\frac{\partial}{\partial x}\left[\rho\left(\frac{\partial\phi}{\partial t}\right)+\frac{\rho}{2}\left[V^2\right]+p\right]=0$$

We can use the y component of the Euler's Equation to find that

$$\frac{\partial}{\partial y}\left[\rho\left(\frac{\partial\phi}{\partial t}\right)+\frac{\rho}{2}\left[V^2\right]+p\right]=0 \tag{3.38}$$

With the term $\left[\rho\left(\frac{\partial\phi}{\partial t}\right)+\frac{\rho}{2}\left[V^2\right]+p\right]$ independent of both x and y as shown in Equations (3.37) and (3.38), we must conclude that $\left[\rho\left(\frac{\partial\phi}{\partial t}\right)+\frac{\rho}{2}\left[V^2\right]+p\right]$ is a constant. We set this constant equal to the total pressure at any time and through the use of the definition of pressure coefficient, we get

$$C_p(\overline{x}_i,\overline{y}_i)=1-\left(\frac{V_{e,i}}{V_\infty}\right)^2-\frac{2}{V_\infty^2}\left(\frac{\partial\phi_i}{\partial t}\right) \tag{3.39}$$

$$\approx 1-\left(\frac{V_{e,i}}{V_\infty}\right)^2-\frac{2c}{V_\infty^2}\left(\frac{\partial V_\infty}{\partial t}\right)$$

The approximation given in Equation (3.39) comes from the conclusion that the dominant term in the time-rate-of-change of the total velocity potential is in proportion to the time-rate-of-change of the freestream velocity. Recall that the total velocity potential is

composed of three components- that due to the freestream velocity, that due to the bound vortex distribution, and finally, the portion due to the distribution of sources and sinks around the airfoil. Now, take two situations, one at time t and a freestream velocity of $V_\infty(t)$ and a second situation at time $t+\Delta t$ and freestream velocity $V_\infty(t+\Delta t)$ where $V_\infty(t+\Delta t)$ is one "velocity unit" great than the previous velocity. Ignore contributions from trailing vortices for the moment. In the first case, the solution produces a specific bound vortex and source/sink distribution. If a solution were required at the new freestream velocity, since Laplace's Equation is linear, the new solution would be in proportion to the ratio of the new freestream velocity to the old one. If this is the case, now all of the velocity potential terms increase by this ratio and thus since the time-rate-of-change of all the terms of the velocity potential follows the freestream velocity, we arrive at the approximation given in Equation (3.39). The chord length, c, in Equation (3.39) is used for dimensional constancy.

The approximation shows that for either large accelerations or for small freestream velocities, the time dependence of the pressure coefficients can be quite large. As will be seen in some of the potential flow solutions, this is true particularly during early portions of the take-off simulation. The resulting pressure coefficients from Equation (3.39) can be integrated to get section lift, drag and moment coefficients as in the steady case. The general result is that an initial acceleration can cause a slight increase in pressure drag and a slight increase in lift due to pressure.

The other modification that is needed in a time-dependent potential flow analysis is a method that handles the conservation of circulation. As a solution is found for each time step, *Kelvin's Theorem* must be upheld. Kelvin's Theorem states that:

> ***Kelvin's Theorem*** *: The circulation around a closed contour which drifts with the fluid is constant with time.*

Prior to motion, there is no circulation in a freestream flowfield in the inviscid analysis. As singularities are added to the flow (line sources/sinks and line vortices from the steady solution) a bound vortex is formed on the airfoil that is the result of an integration of the line vortex around the surface of the airfoil. While an integration of the source/sink distribution gives no net mass added to the system, in order to create the lift and satisfy the Kutta Condition, we arrive at a bound vortex of strength Γ_{bound} given by:

$$\Gamma = \oint \gamma \, dl \tag{3.40}$$

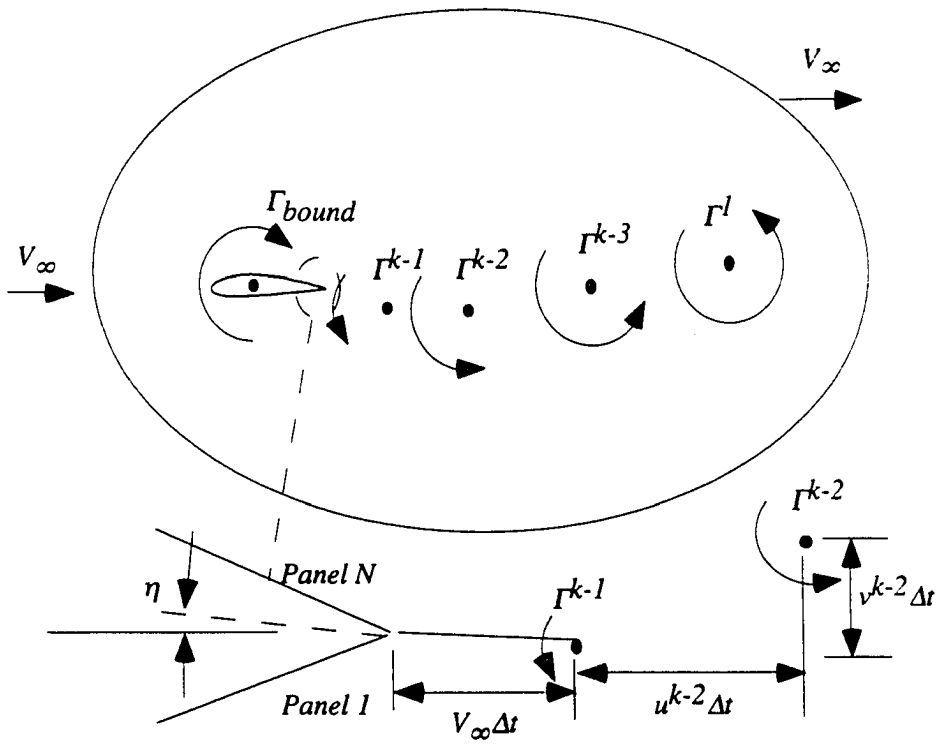


Figure 3.6: Drifting away of the starting vortices and transport of the vortices

Figure 3.6 shows Kelvin's theorem in operation. Since no circulation is present in a flowfield for an airfoil that starts at rest, the sum of all circulation must always be zero. At time step k, for example, there is a bound vortex, Γ_{bound} (positively defined Γ is in the clockwise direction). Previously, at time step k-1, a vortex was shed from the airfoil so as to conserve circulation. Note that the superscript, k-1, on Γ^{k-1} corresponds to the vortex that is shed at time step k-1. Likewise, the superscript, k-2, on Γ^{k-2} corresponds to the vortex shed at time step k-2.

The manner in which it is shed is also shown in Figure 3.6. It is assumed that the vortex is convected at the freestream velocity during the time increment, Δt, and travels a distance parallel to the bisector of the trailing edge shown by the angle η in Figure 3.6. Actually, this is a possible point of contention. The physics of the problem are debatable in this region. Even during a small time period, Δt, a vortex which is shed moves from the trailing edge which is considered to be a stagnation point to a point in the flow field at which there is a finite velocity. The decision was made to use the freestream velocity as the

convecting velocity during this time period after several tests allowing the convecting velocity to be anywhere from the tangential velocity at the last panels (a minimum) up to the freestream velocity (a maximum). No qualitative differences were found on the section lift and drag coefficients under these tests. This decision was made and used consistently.

Once the point vortex passes through this initial time period it will travel a distance $u^{k-2}\Delta t$ in the horizontal direction and a distance $v^{k-2}\Delta t$ in the vertical direction where u^{k-2} and v^{k-2} are determined from the global velocity field at that point. The global velocity at a point is determined from the sum of the freestream velocity and the contribution from all the singularities in the flowfield including the line source/sink distribution and the line vortex distribution on the airfoil. Also contributing to the global velocity at that point are the other vortices that have been shed.

The most important feature of these shed vortices is in the way that they modify the no penetration condition and the Kutta Condition for each new solution. Suppose at time step $k+1$, we are interested in finding the pressure distribution on the airfoil. Now, rather than just the line source/sink and line vortex distribution on the airfoil contributing normal velocities at each panel and creating a tangential velocity that is matched on the upper and lower surfaces at the trailing edge panels, there are contributions from all of the shed vortices as well.

Figure 3.7 shows the geometry of the velocity contribution at each panel. Suppose there is a shed vortex of strength Γ^m from time step m at a global position of (x_m, y_m). If

$$\hat{\theta}_m = \tan^{-1}\left[\frac{\bar{y}_i - y_m}{\bar{x}_i - x_m}\right] \tag{3.41}$$

where $\hat{\theta}_m$ is the angle the vector $r_{mi}\hat{e}_{r,m}$ (the one that points from the origin of the shed vortex at time step m to the midpoint of panel i) makes with the global x coordinate positive direction and

$$\underline{V}_{\Gamma^m} = \frac{-\Gamma^m}{2\pi r_{mi}}\hat{e}_{\hat{\theta},m} \tag{3.42}$$

Then, with

$$\begin{aligned} \hat{n}_i &= -\sin\theta_i\,\hat{i} + \cos\theta_i\,\hat{j} \\ \hat{t}_i &= \cos\theta_i\,\hat{i} + \sin\theta_i\,\hat{j} \end{aligned} \tag{3.43}$$

being normal and tangential unit vectors for panel i and

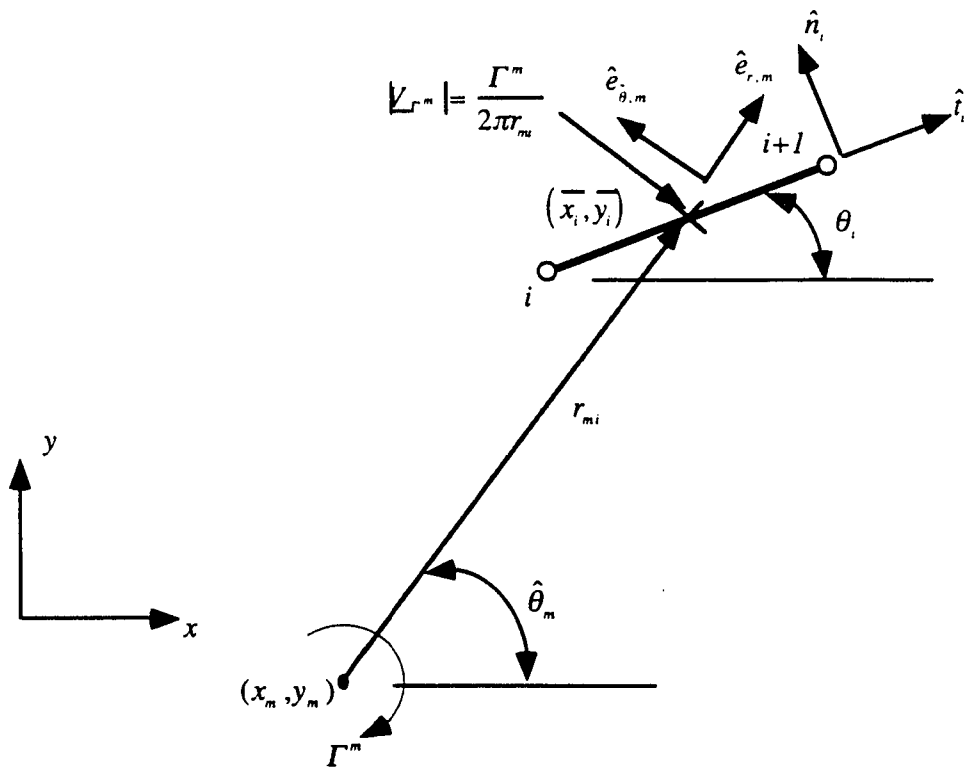


Figure 3.7: No penetration condition and tangential velocity modification by trailing vortices

$$\hat{e}_{\hat{\theta},m} = -\sin\hat{\theta}_m\,\hat{i} + \cos\hat{\theta}_m\,\hat{j} \tag{3.44}$$

the velocity induced at the midpoint of panel j by a vortex at (x_m, y_m) can be written in normal and tangential coordinates as

$$\begin{aligned} \underline{V}_{\Gamma^m} &= \frac{-\Gamma^m}{2\pi r_{mi}}\hat{e}_{\hat{\theta},m} \\ &= V_{tangential}\hat{t}_i + V_{normal}\hat{n}_i \\ &= \frac{\Gamma^m}{2\pi r_{mi}}\sin(\hat{\theta}_m - \theta_i)\hat{t}_i - \frac{\Gamma^m}{2\pi r_{mi}}\cos(\hat{\theta}_m - \theta_i)\hat{n}_i \end{aligned} \tag{3.45}$$

Now, in modifying the solution to the source/sink distribution along the panels, we must add the contribution at each panel due to the vortices. Fortunately, the only requirement is to modify the b column vector (See Equation (3.21)) that accounts for the freestream

contribution to the normal velocity. To each element of the b vector (see Equation (3.24), we add such that:

$$b_{i,modified} = b_{i,steady} + \sum_{m=1}^{k-1} \frac{\Gamma^m}{2\pi r_{mi}} \cos(\hat{\theta}_m - \theta_i) \text{ for } i = 1,..., N \tag{3.46}$$

Notice the sign change. If there is an induced velocity at the panel, we want the other sources to counteract this normal velocity to make the total sum of normal velocity equal to zero.

Now, element N+1 in column b (see Equations (3.25) and (3.28)) gets the additional tangential velocity contributions from the vortices

$$b_{N+1,modified} = b_{N+1,steady} + \sum_{m=1}^{k-1} \sum_{i=1,N} -\frac{\Gamma^m}{2\pi r_{mi}} \sin(\hat{\theta}_m - \theta_i) \tag{3.47}$$

Ultimately, when the tangential velocities are calculated on each panel, we must include the contribution from each of the vortices.

Thus, the time-dependent modification of the panel method involves several steps:

1. Account for the conservation of circulation via Kelvin's Theorem.
2. Modify the boundary conditions of no penetration and the Kutta condition at the trailing edge due to the trailing vortices and
3. Account for the contribution to the tangential velocities due to the trailing vortices in the calculation of the pressure coefficients.

One last item that deserves mention is that in this scheme, the time rate of change of the velocity potential at any given point in the flowfield is approximated by

$$\frac{\partial \phi}{\partial t} = \frac{\phi^{k+1} - \phi^k}{\Delta t} \tag{3.48}$$

This requires knowledge of all of the various velocity potentials including those contributed by the line source/sink distributions, the line vortex distributions and the trailing vortices.

3.6 Typical potential flow solutions

At this point, it is useful to look at some of the potential flow solutions acquired in this study. As a matter of consistency, we will inspect the solution for a NACA 0012 airfoil section. This airfoil section has numerous experimental data available for verification, and will instill confidence in the solutions obtained. Unless otherwise stated, the airfoil will be the

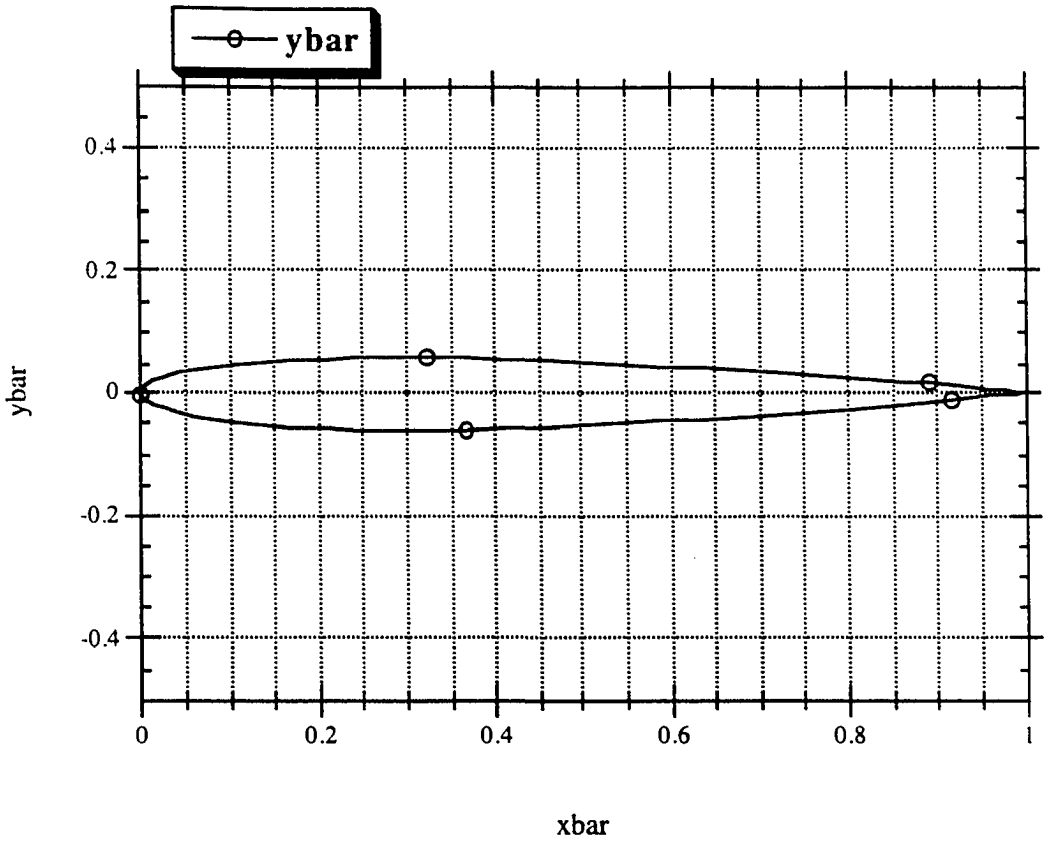


Figure 3.8: Sketch of NACA 0012 airfoil

NACA 0012 section with 35 nodes on top of the airfoil and 35 nodes on the bottom of the airfoil. A sketch of the NACA 0012 airfoil used here is shown in Figure 3.8.

First of all, without respect to speed, acceleration or viscous effects, the predicted steady state pressure distribution solutions for several angles of attack appear in Figure 3.9. One of the key points in reading a pressure distribution graph like Figure 3.9 is to realize that it is the pressure *differential* between the top and bottom surfaces that generate lift. Both the pressures on the upper and lower surfaces of the airfoil are shown in Figure 3.9. These designations will be omitted on subsequent graphs. Generally speaking, as the angle of attack increases, you can see the differential pressure between the upper and lower surface widen at

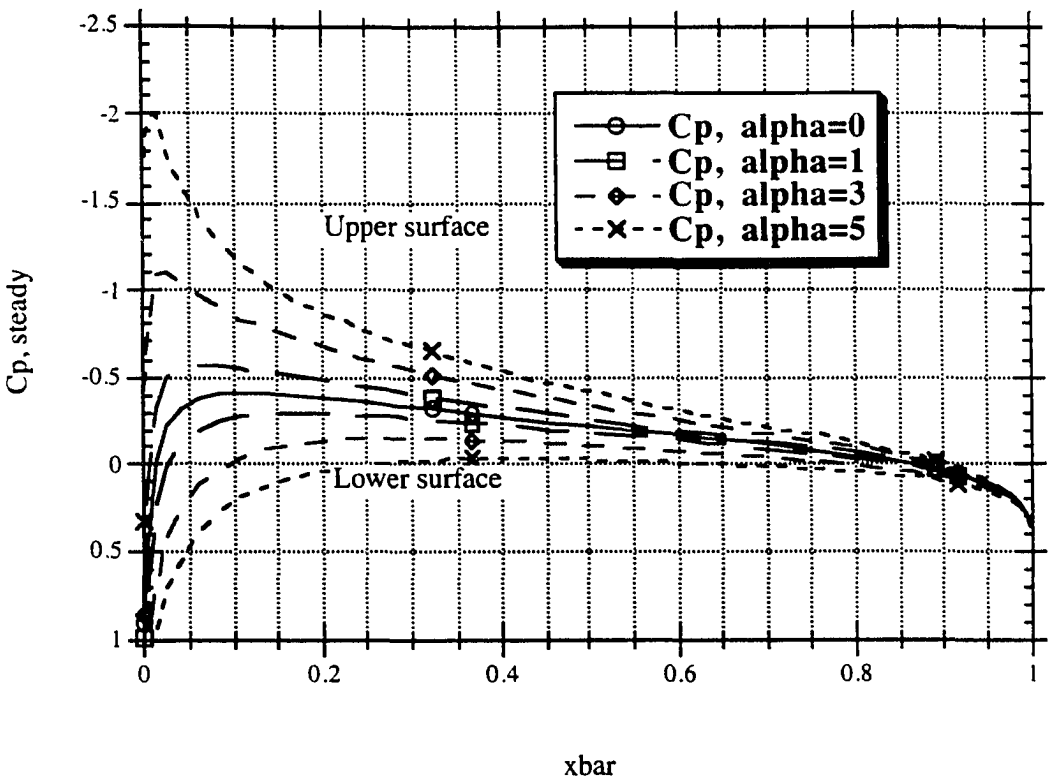


Figure 3.9: Pressure distributions on a NACA 0012 airfoil for steady flow at 0, 1, 3, and 5 degrees angle of attack

each chordwise station. The highest negative values on the graph (plotted upward out of convention) are indicative of regions of suction. In fact, any point, even on the lower surface that has a negative pressure coefficient is known to produce a pressure that is less than the ambient, or reference pressure.

For varying angles of attack, there is an expected linear increase in lift. In the steady flow pressure distributions shown in the figure, the section lift coefficient estimates rise from 0.0000 at 0° angle of attack (AOA) to 0.6012 at 5° AOA. See Table 3.1.

Table 3.1: Potential flow results for a NACA 0012 airfoil- no acceleration

Angle of attack (degrees)	Section lift coefficient (c_l)	Section drag coefficient (c_d)
0.0	0.0000	0.0005
1.0	0.1204	0.0005
3.0	0.3610	0.0005
5.0	0.6012	0.0007

In the analysis of a take-off simulation, as mentioned above, there are time-dependencies that must be accounted for. At a nominal initial angle of attack of 1°, the steady flow section lift coefficient is 0.1204 for this airfoil. When the airfoil is accelerated at a rate of 2.5 m/s^2 (here we will assume that the chord length is 1.0 meter long and that units of velocity are in m/s with units of linear acceleration in m/s^2) with an initial velocity of 3.5 m/s the resulting pressure distributions appear as in Figure 3.10. The predicted lift coefficient for this accelerated flow is 0.1277. As the freestream velocity increases, this difference is not as great. The section lift coefficient approaches the steady state values. A feature of potential flow analysis is that in the steady case, as the number of nodes is increased, the section drag coefficient approaches 0.0. This is a result of not accounting for the viscosity. On the other hand, including the acceleration terms, there is a drag coefficient that is due to pressure. In the situation shown in Figure 3.10, the section drag coefficient is estimated to be 0.0235- at this point, entirely due to the pressure distribution.

3.7 Take-off simulation

In the analysis of the effect of deicing fluid on a general aviation airfoil under take-off conditions, there are several parameters to consider. Figure 3.11 shows these as applied to the simulation at hand. Within this work the initial velocity must be specified. In Figure 3.11, this appears as 3.5 m/s. The initial velocity is important as the airfoil is allowed to develop a "steady-state" prior to acceleration. The low-speed initial velocity has a series of vortices that are initially shed to conserve circulation. The sum of all of the shed vortices during this time period is equal to the bound vortex that results in the steady-state lift coefficient prior to acceleration.

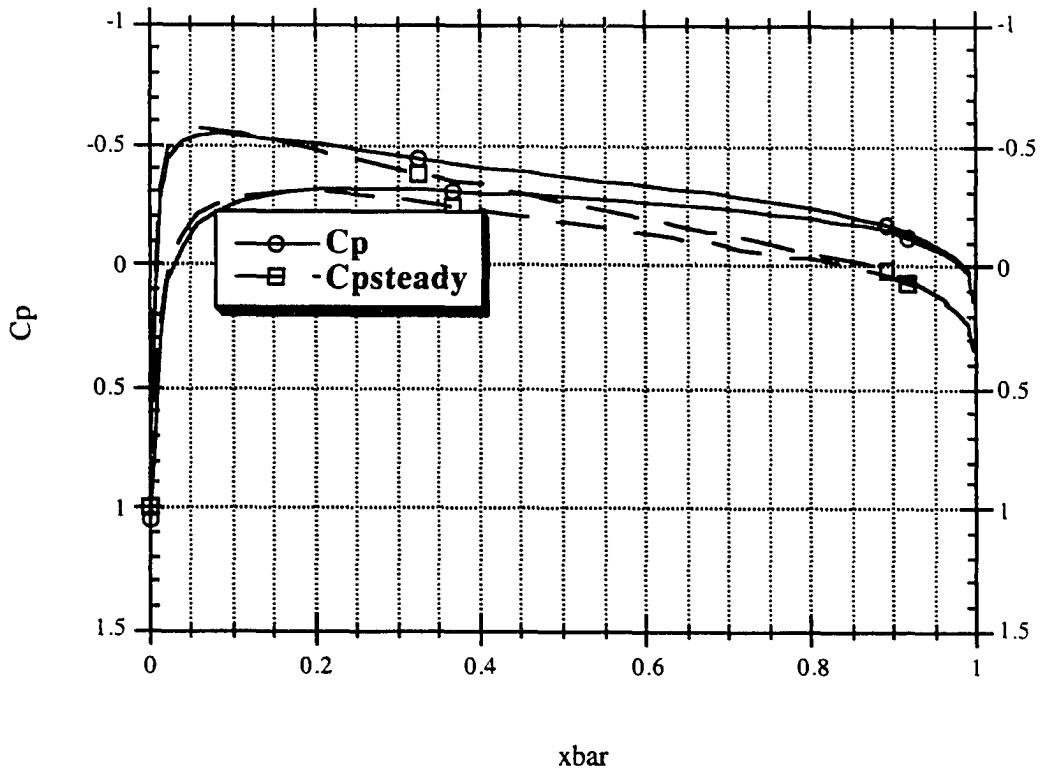


Figure 3.10: Effect of an acceleration on potential flow pressure distribution at low speeds

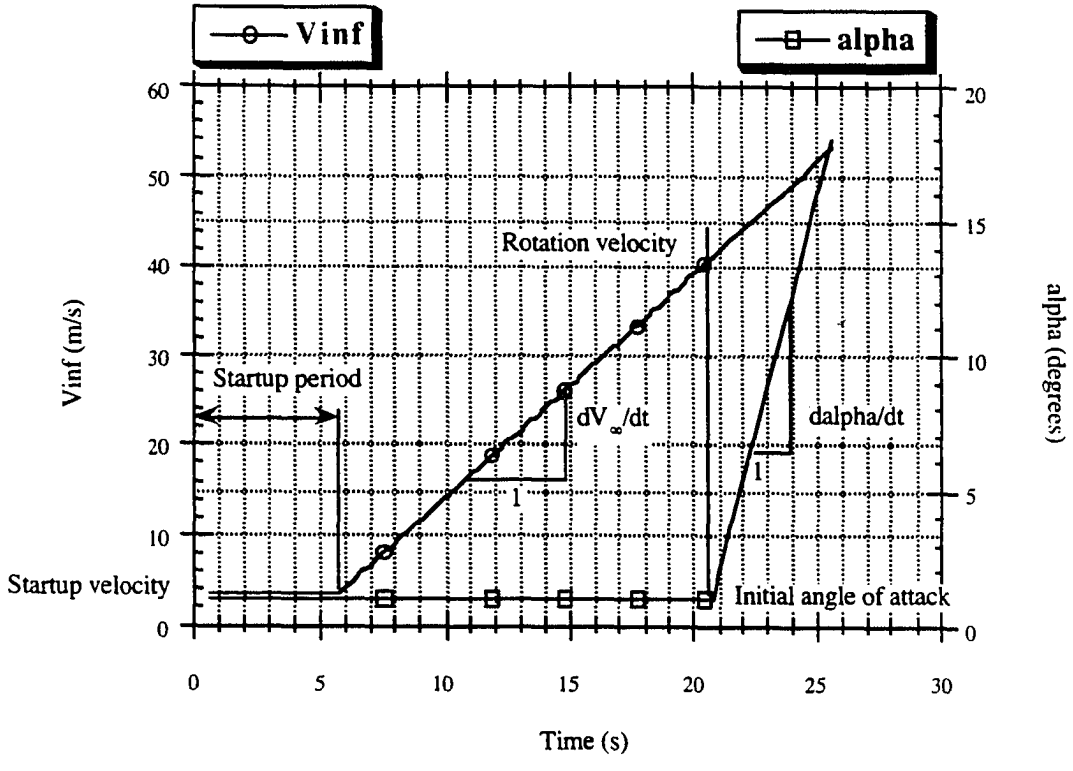


Figure 3.11: Take-off simulation control parameters

Another parameter which is deemed important in this study is the initial angle of attack, α_0. As shown in Figure 3.11, the initial angle of attack is 1.0°. This value can change, certainly, depending on the aircraft under study and in fact usually varies across the span, but in general, these values usually range from about 0.0° to maybe 3.0°. Thus, the initial angle of attack is important. It is also thought that this will be more critical as the deicing fluid is placed on the wing. Varying initial angles of attack will give rise to different pressure distributions and resulting shear stress distributions which could certainly effect the flow of the deicing fluid differently.

After the steady state is found at the initial velocity (by requiring that the section lift, drag, and moment coefficients don't change "much" from one time step to the next), the airfoil is allowed to accelerate. This is the third parameter of interest. As shown in Figure 3.11, the acceleration of the freestream, dV_∞ / dt, in this case, is set at 2.5 m/s^2. This is typical of general aviation aircraft. A propeller-piston engine method of propulsion usually provides an acceleration which is nearly independent of speed.

The next parameter of interest will be the rotation speed, V_r. Once the airfoil is accelerated, at what point does the angle of attack begin to increase per a rotation maneuver? The smallest aircraft generally have the smallest rotation speeds while the larger aircraft generally have larger rotation speeds. The longer the take-off run, i.e. the larger the rotation speed, the more chance there is for the fluid to flow off the wing. The rotation speed is shown as 41.0 m/s in Figure 3.11.

Finally, the rate at which the angle of attack increases in the rotation maneuver, $d\alpha / dt$, will be important. This adds a time-dependent component to the flowfield. The pitch rate in Figure 3.11 is shown to be 3.5°/second.

With the parameters defined as in Figure 3.11, the resulting potential flow analysis for section lift and drag coefficients appears in Figure 3.12. Several features are apparent. Shown in Figure 3.12 are two different section lift coefficient curves. The first, $c_{l,steady}$, is the section lift coefficient if only the tangential velocities are taken into account (see Equation (3.30)). The second, c_l, takes the effects of acceleration into account (see Equation (3.39)). As stated before, the difference, at low speeds, is to slightly increase the expected lift coefficient at the expense of increasing drag. This analysis is typical of the potential flow solution results for the take-off simulation. The next step is to account for viscous effects in the gas-dynamic boundary layer.

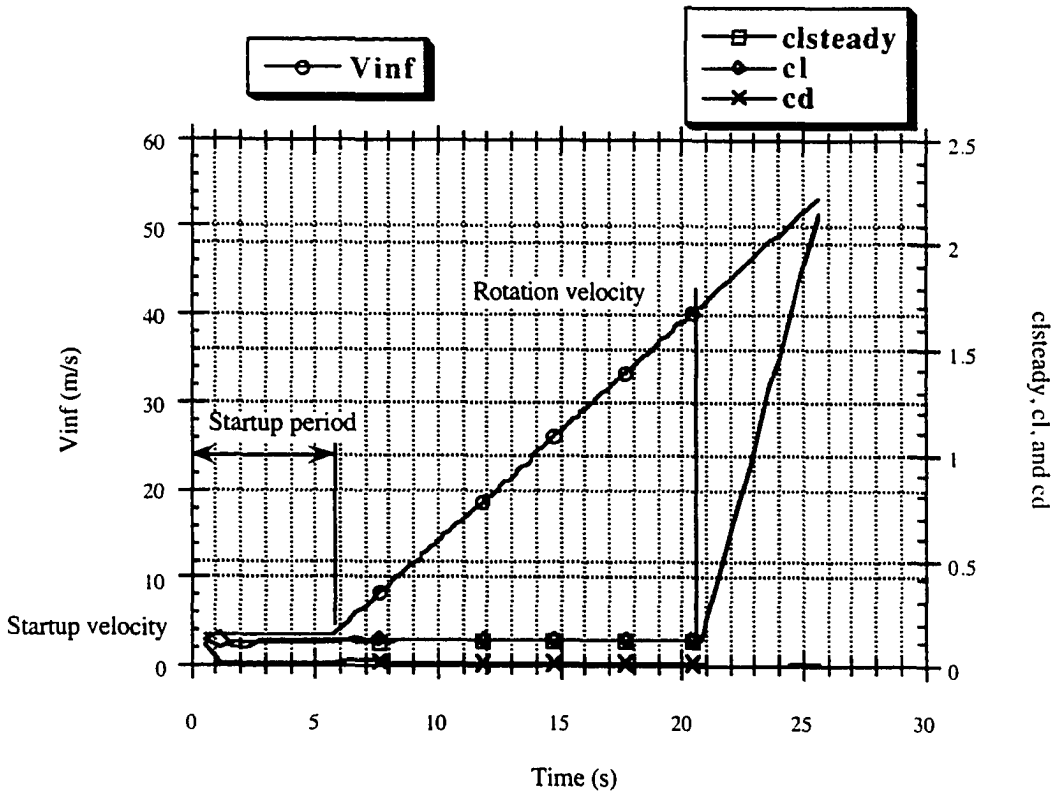


Figure 3.12: Results of a typical potential flow analysis

4. METHODS OF ANALYSIS: THE GAS-DYNAMIC BOUNDARY LAYER

4.1 Development of the boundary layer equations

To this point, no intent has been made to account for viscous effects of the air passing the airfoil expect for the very real consequence of the Kutta Condition in the analysis of the potential flow in two dimensions.

Behaving as a real gas, the air is governed by the most general equations of motion, the conservation of momentum equations given in Equation (4.1). The assumption in this set of equations is that the fluid is incompressible and the shear stress-rate of strain relationship in the fluid is linear.

$$\rho\frac{D\underline{V}}{Dt} = -\nabla p + \rho\underline{g} + \mu\nabla^2\underline{V} \tag{4.1}$$

Here, $\underline{V}$ is the velocity vector, p is the pressure, $\underline{g}$ is the acceleration due to gravity, ρ is the fluid density, and μ is the dynamic viscosity of the fluid. Along with the conservation of mass equation for an incompressible fluid,

$$\nabla\cdot\underline{V} = 0 \tag{4.2}$$

we have a set of equations that govern the motion of the real fluid. These are known collectively as the Navier-Stokes equations.

In two dimensional Cartesian coordinates (x and y), these equations become

$$\rho\frac{Du}{Dt} = \rho\left(\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y}\right) = -\frac{\partial p}{\partial x} + \rho g_x + \mu\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) \tag{4.3}$$

$$\rho\frac{Dv}{Dt} = \rho\left(\frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y}\right) = -\frac{\partial p}{\partial y} + \rho g_y + \mu\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right) \tag{4.4}$$

where u and v are the components of velocity parallel and perpendicular to the airfoil surface. The conservation of mass equation gives:

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0 \tag{4.5}$$

In the engineering analysis of the effect of the deicing fluid on the aerodynamic performance of the airfoil, we will find that certain assumptions are appropriate. While this set of equations is assumed to be valid throughout the flowfield, there is a region, known as the boundary layer, or in this case, the gas-dynamic boundary layer, defined as a region in which viscous effects are important. This region is known to be in a very small area near the airfoil. The first assumption that is made in this boundary layer region is that the vertical component of velocity is small in comparison to the horizontal component.

$$v << u \tag{4.6}$$

The second assumption is that the gradient in the streamwise direction is much less than that in the normal direction.

$$\frac{\partial}{\partial x} << \frac{\partial}{\partial y} \tag{4.7}$$

With these two assumptions in place, and neglecting the effect of gravity, an order of magnitude analysis reveals the following set of equations, known as the Boundary Layer Equations:

$$\rho\left(\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y}\right) = -\frac{\partial p}{\partial x} + \mu\left(\frac{\partial^2 u}{\partial y^2}\right) \tag{4.8}$$

$$0 = \frac{\partial p}{\partial y} \tag{4.9}$$

and

$$\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0 \tag{4.10}$$

Inherent in these equations is the neglect of any normal acceleration of the fluid; therefore motion from one station to the next is assumed to occur over a region that is completely flat. Figure 4.1 shows the general coordinate system that is used in the solution of the gas-dynamic boundary layer. The airfoil has a stagnation point that is defined by the potential flow at any time. Proceeding along the upper surface of the airfoil, the edge velocities are known outside the boundary layer, δ, and we proceed for positive values of x till the trailing edge. Perpendicular to this is the y coordinate. The negative x coordinate runs along the lower surface to the trailing edge with boundary layer thicknesses, δ, again. Shown in

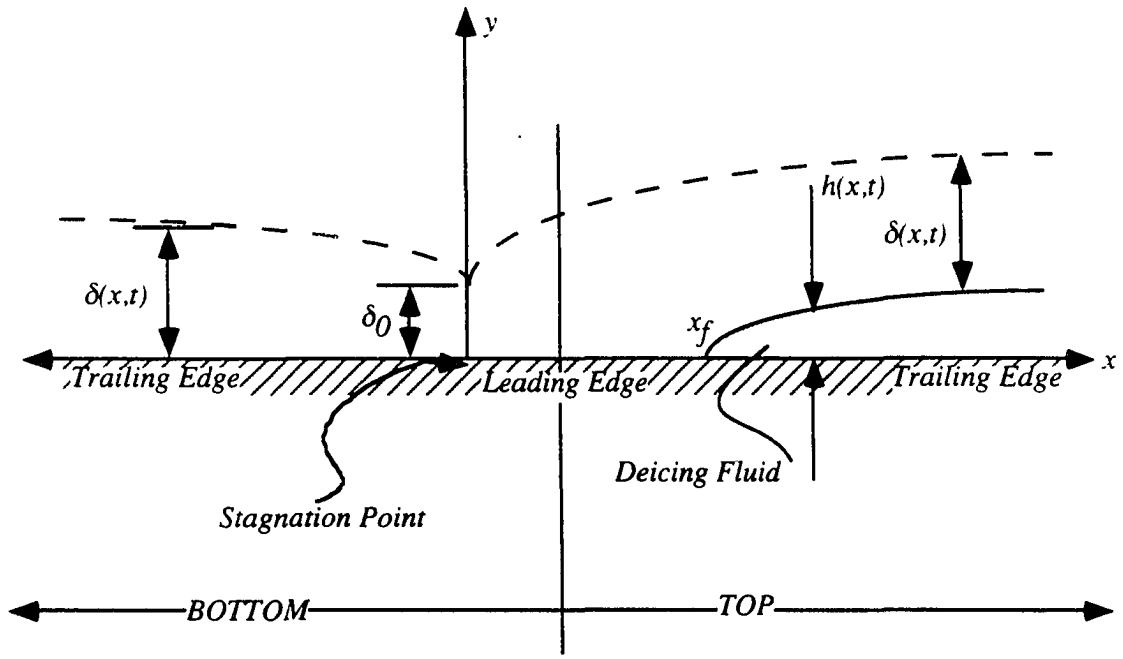


Figure 4.1: Boundary layer coordinate system

Figure 4.1 is the deicing fluid layer which begins at the coordinate x_f. Details of the deicing fluid will be explained in the next Chapter.

As can be seen in Equation (4.9), the pressure variation along the vertical direction is zero. If this is true, the pressure outside the boundary layer is assumed to the pressure inside the boundary layer. This is realized in Euler's approximation to the time-dependent pressure gradient, namely:

$$-\frac{\partial p}{\partial x} = \rho\left(\frac{\partial V_e}{\partial t} + V_e \frac{\partial V_e}{\partial x}\right) \tag{4.11}$$

where V_e is the tangential velocity at a point, x, on the airfoil as determined by the potential flow solution that depends on time. There yet remains a distinction between laminar and turbulent flow. In turbulent flow, we will see that there is an additional component of shear stress that must be added to the right-hand side of Equation (4.8). For now, let us ignore this contribution.

There are several boundary conditions that are appropriate to the solution of Equation (4.12) (rewritten for clarity) that will give a unique solution to the flowfield variables of u and v.

$$\rho\left(\frac{\partial u}{\partial t}+u\frac{\partial u}{\partial x}+v\frac{\partial u}{\partial y}\right)=\rho\left(\frac{\partial V_e}{\partial t}+V_e\frac{\partial V_e}{\partial x}\right)+\mu\left(\frac{\partial^2 u}{\partial y^2}\right) \tag{4.12}$$

Equation (4.12) requires one initial condition for the horizontal component of velocity and two boundary conditions in space. The general procedure is to solve for the u values at any station and use the conservation of mass equation to solve for v. The boundary conditions for u are the "no slip" condition and the fact that the horizontal component of velocity approach the freestream value of tangential velocity at that station at the edge of the boundary layer. In fact, the boundary layer thickness is defined as the point at which the horizontal component of velocity is 99% of the freestream or edge velocity at that station. When there is no deicing fluid present, the no slip condition says that the total velocity (both components) is zero at the bottom of the boundary layer. This is modified if the deicing fluid is present. As will be mentioned later, the boundary condition on the gas-dynamic boundary layer is then such that the tangential velocity at the interface of the deicing fluid and the gas-dynamic boundary layer are equivalent for both the gas-dynamic boundary layer and the deicing fluid. For the moment, the lower boundary condition is set as a zero velocity

4.2 Turbulence

As the gas-dynamic boundary layer develops around the airfoil, it is a well-known fact that at some point the flow makes a transition from laminar to turbulent flow. The main distinction between these two flow regimes is that in laminar flow, the relationship between shear stress and rate of strain is linear (the constant of proportionality given by the dynamic viscosity) while in turbulent flow, this relationship is not always true.

In a turbulent flow, there is an attempt made to model the effect of perturbations on mean parameters such as velocity and pressure. These primitive variables can be thought to be made up a mean value and a perturbation as in Equation (4.13).

$$p=\bar{p}+p' \tag{4.13}$$

$$u=\bar{u}+u'$$

$$v=\bar{v}+v'$$

Here, the mean value of a variable is given by

$$\bar{u} = \lim_{T\to\infty} \frac{1}{T} \int_{t_0}^{t_0+T} u\, dt \tag{4.14}$$

where u' is usually considered to be much less than $\bar{u}$. A process of "time-averaging" exists such that for variables a and b: where $\overline{(\ )}$ means to take the time-average of $(\ )$,

$$\begin{aligned} \overline{a+b} &= \bar{a} + \bar{b} \\ \overline{a'} &= 0 \\ \overline{\frac{\partial a}{\partial x}} &= \frac{\partial \bar{a}}{\partial x}, \overline{\frac{\partial a}{\partial y}} = \frac{\partial \bar{a}}{\partial y} \\ \overline{\bar{a}b} &= \bar{a}\bar{b} \end{aligned} \tag{4.15}$$

but

$$\begin{aligned} \overline{ab} &= \overline{(a+a')(b+b')} \\ &= \overline{\bar{a}\bar{b}} + \overline{a'\bar{b}} + \overline{\bar{a}b'} + \overline{a'b'} \\ &= \bar{a}\bar{b} + \overline{a'b'} \end{aligned} \tag{4.16}$$

Note that even though the averages of a' and b' are zero, the average of their product, $a'b'$ need not be. Replacing each of the primitive variables, u, v, and p by their representations in Equation (4.13) and taking a time average of Equation (4.12) gives rise to an additional term that was not present before. This is the so-called *Reynolds Stress* term, $\frac{\partial}{\partial y}\left[-\overline{\rho u'v'}\right]$. Thus,

$$\rho\left(\frac{\partial \bar{u}}{\partial t} + \bar{u}\frac{\partial \bar{u}}{\partial x} + \bar{v}\frac{\partial \bar{u}}{\partial y}\right) = -\frac{\partial \bar{p}}{\partial x} + \frac{\partial}{\partial y}\left[\mu\left(\frac{\partial u}{\partial y}\right) - \overline{\rho u'v'}\right] \tag{4.17}$$

or

$$\rho\left(\frac{\partial \bar{u}}{\partial t} + \bar{u}\frac{\partial \bar{u}}{\partial x} + \bar{v}\frac{\partial \bar{u}}{\partial y}\right) = \rho\left(\frac{\partial V_e}{\partial t} + V_e\frac{\partial V_e}{\partial x}\right) + \frac{\partial}{\partial y}\left[\mu\left(\frac{\partial u}{\partial y}\right) - \overline{\rho u'v'}\right] \tag{4.18}$$

Now, there are no known mathematical methods to truly model this Reynolds Stress term. There have been entire careers based on formulating the best models for this term. A variety of methods have been used successfully to model this term, all having a base in empirical data. There are several choices at this point. Turbulence models based on algebraic information, so-called first-order methods that depend on a single ordinary differential

equation, second-order methods based on partial differential equation relationships between mean values of the primitive variables and others are among the options.

It should be pointed out that the conservation of mass equation that goes with this conservation of momentum equation remains similar under the time averaging process. Namely:

$$\frac{\partial \bar{u}}{\partial x} + \frac{\partial \bar{v}}{\partial y} = 0 \tag{4.19}$$

4.2.1 Algebraic models

Perhaps a point of discussion and potential improvement in later work, is the turbulence modeling used in this study, but an algebraic model was chosen for simplicity. In this study, a method known as Prandtl's mixing length was used.

In the Prandtl's mixing length method of turbulence modeling, the shear stress term, $-\overline{\rho u'v'}$, is said to be proportional to a mixing length, l_m such that:

$$\tau_T = -\overline{\rho u'v'} = \rho l_m^2 \left|\frac{\partial \bar{u}}{\partial y}\right| \frac{\partial \bar{u}}{\partial y} \tag{4.20}$$

The key is in how to model the mixing length, l_m. One of the most successful efforts was made by Van Driest [41]. His model assumes the mixing length varies depending on the region of the boundary layer. Within the turbulent boundary layer are regions known as a laminar region in which the effects of turbulence are minimal, a buffer zone that is a transition from the laminar region to the fully turbulent region to, finally, an outer region. See Figure 4.2. The graph in Figure 4.2 shows a non-dimensional velocity , u^+, plotted as a function of a non-dimensional distance, y^+ where:

$$u^+ = \frac{\bar{u}}{u^*} \tag{4.21}$$

$$y^+ = \frac{yu^*}{\nu} \tag{4.22}$$

and

$$u^* = \sqrt{\frac{|\tau_w|}{\rho}} \tag{4.23}$$

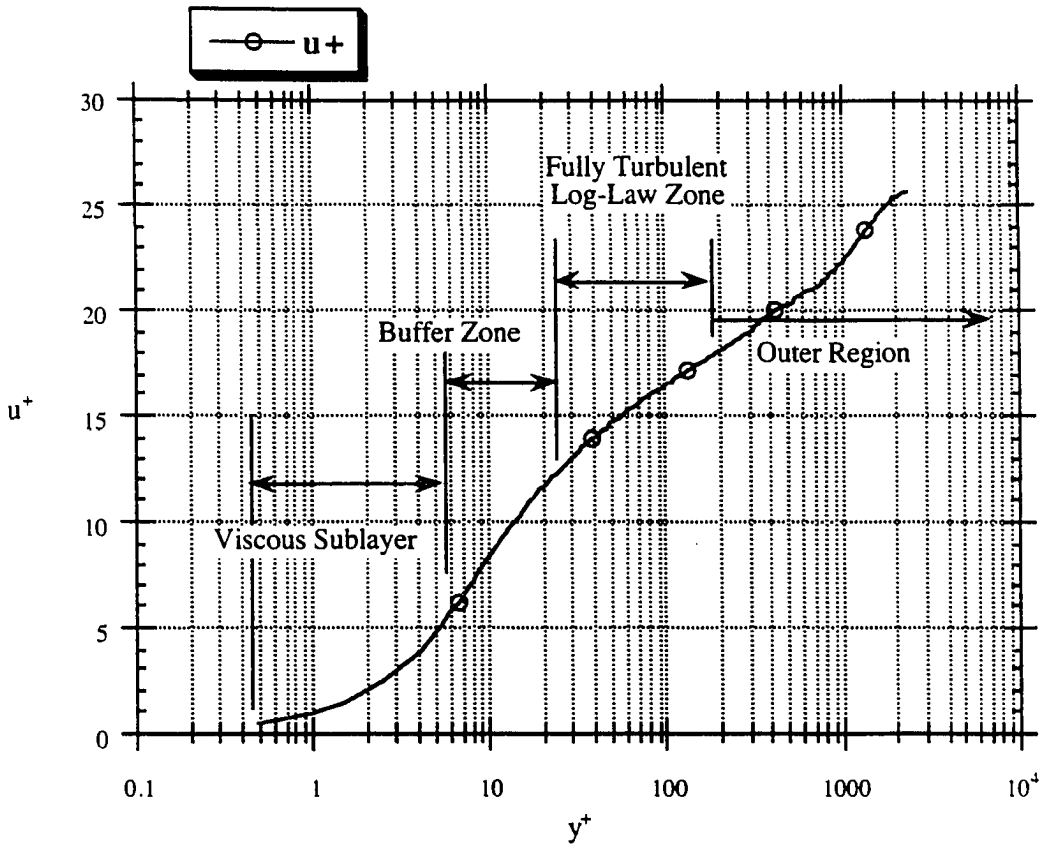


Figure 4.2: Typical turbulent boundary layer profile

Here, ν is the kinematic viscosity of the fluid. The term, u^* is known as the friction velocity. For a large set of empirical data, the curves are very similar to that shown in Figure 4.2. Van Driest chose his model of the mixing length as to closely match the shape of the turbulent boundary layer. In the Van Driest model, the mixing length is defined such that there are two regions of interest. Within the "inner" layer, the mixing length is assumed to follow:

$$l_{m,inner} = \kappa y\left(1 - e^{y^*/A^*}\right) \tag{4.24}$$

where κ is known as the von Kármán constant usually taken to be 0.41 and A^+ is a damping constant usually taken to be 26. Within the "outer" region, the mixing length is approximated by

$$l_{m,outer} = C_l \delta \tag{4.25}$$

where δ is the boundary layer thickness and C_l usually assigned a value of 0.089. Once the value predicted by the inner model exceeds this value, the switch is made to the outer model.

4.2.2 Corrections for pressure gradients and low Reynolds numbers

Van Driest based his constant values on data obtained for flat plate experiments (ones that did not included pressure gradient effects). There are several recommendations that have been made to include the effects of pressure gradient and low Reynolds number flow. The following two modifications have been used in this model. First, to account for pressure gradient, White [41] suggests the following modification to the damping constant, A^+.

$$A^+ = \frac{26}{\left(1 + bp^+\right)^{1/2}} \tag{4.26}$$

where

$$p^+ = \frac{\nu}{\rho\left(u^*\right)^3}\frac{\partial p}{\partial x} \tag{4.27}$$

and $b = 12.6$ if $p^+ > 0$ and $b = 14.76$ if $p^+ < 0$. Another modification is suggested by Anderson, Tannehill and Pletcher [1] that takes into account "low Reynolds numbers". Predictions can be brought into good agreement with measurements at low Reynolds numbers by simply delaying the switch from the inner model (Equation (4.24)) to the outer (Equation (4.25)) until $y^+ \geq 50$. If, at $y^+ = 50$ in the flow, $l_m/\delta \leq 0.089$, no adjustment is necessary. On the other hand, if the inner model predicts $l_m/\delta > 0.089$, then the mixing length becomes constant in the outer region at the value computed at $y^+ = 50$. They claim this simple adjustment ensures the existence of the log-linear region in the flow which is agreement with the preponderance of measurements.

Both of these modifications were used in adjusting the mixing length parameter in this method of turbulent shear stress modeling.

4.3 Transition

The next question is, at what point does the transition occur? Transition to turbulence is really a process that happens over a finite length of the airfoil. In this study, however, no attempt was made to model the transition process, moving from an instability to the fully turbulent flow; rather, the flow was either set to laminar flow, or allowed to be turbulent where the turbulent shear stresses were included in the model equation. So, this study does not allow for the process of transition, only the existence of turbulence. The final result of this is that transition was based on experimental data that is available in the literature.

Figure 4.3 [41] shows the transition point for an accumulation of data where the flow is expected to change to turbulence. If the Reynolds number based on displacement thickness is large enough for a given shape factor, H, then the flow is deemed critical and turbulent shear stresses are included in the model beyond that point. Within a boundary layer, there are several terms of interest. The first is the displacement thickness, δ^*, where

$$\delta^*(x,t) = \int_0^{\infty}\left(1-\frac{\bar{u}(x,t)}{V_e(x,t)}\right)dy = \int_0^{\delta}\left(1-\frac{\bar{u}(x,t)}{V_e(x,t)}\right)dy \tag{4.28}$$

and the second is the momentum thickness

$$\theta(x,t) = \int_0^{\infty}\left(1-\frac{\bar{u}(x,t)}{V_e(x,t)}\right)\left(\frac{\bar{u}(x,t)}{V_e(x,t)}\right)dy = \int_0^{\delta}\left(1-\frac{\bar{u}(x,t)}{V_e(x,t)}\right)\left(\frac{\bar{u}(x,t)}{V_e(x,t)}\right)dy \tag{4.29}$$

The ratio of the displacement thickness to the momentum thickness is the shape factor,

$$H(x,t) = \frac{\delta^*(x,t)}{\theta(x,t)} \tag{4.30}$$

The data shown in Figure 4.3 account for other factors such as the pressure gradient in the form of the shape factor. It should be noted that the Reynolds number based on displacement thickness is defined locally such that

$$Re_{\delta^*} = \frac{\rho V_e \delta^*}{\mu} \tag{4.31}$$

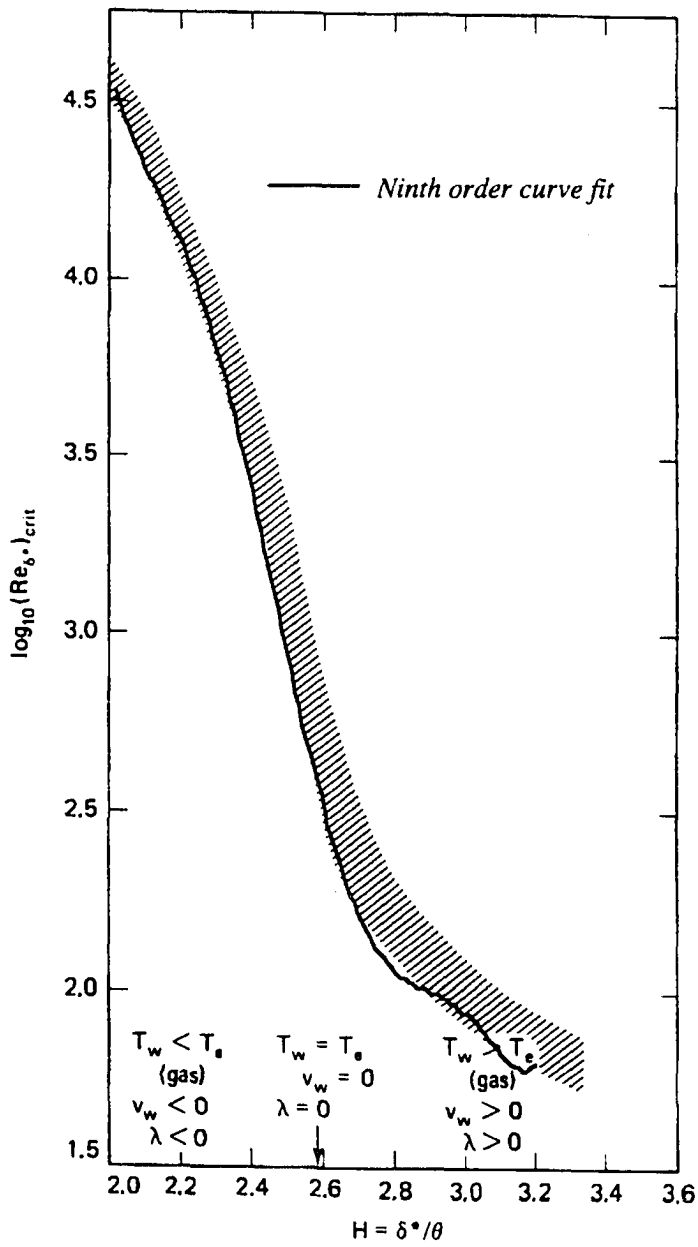


Figure 4.3: Experimental data on transition

As shown in Figure 4.3 there is some scatter in the data. For the purposes of this study, a consistent model was used in that a curve fit of the representative data was used to predict transition. The figure was scanned into a "PICT" file, and using a program called "Data_Thief" for the Macintosh operating system, the axes were defined and a series of data points was "stolen". This data was sent to a file and the resulting curve fit was used.

$$\log_{10}(\mathrm{Re}_{\delta^*})_{critical} = \sum_{i=1}^{10} a(i) H^{i-1} \tag{4.32}$$

$$a1 = 554.884547077422$$
$$a2 = -305.637315183644$$
$$a3 = -400.403765708399$$
$$a4 = 137.244271837635$$
$$a5 = 327.134839498025$$
$$a6 = -258.649002578856$$
$$a7 = 59.5446175847718$$
$$a8 = 3.54613361353859$$
$$a9 = -3.27161696528034$$
$$a10 = 0.351442474509252$$

Also shown in Figure 4.3 is the result of the curve fit. It is represented by the solid line in the figure which overlays the graph taken from the reference. You may have to look carefully, but the fit follows the lower bound of the data scatter and for the range shown, is fairly free of the oscillations that can be present with high-order approximations. For purposes of the programming, the range of the shape factor where the curve fit was applicable (and transition was allowed) was between 2.00 and 3.10. Most of the transitions in this study occurred near a shape factor of about 2.55. This is, in fact the area where the classical flat-plate problem reaches transition. The similar solution has a shape factor of approximately 2.6.

4.4 Stagnation point flow

As can be seen in Equation (4.12), there needs to be some initial condition from which to start the solution. The method chosen in this study was to assume that at the stagnation point, the freestream velocity varied linearly from the midpoint of one panel to the next. Figure 4.4 shows some of the characteristics of the stagnation point flow.

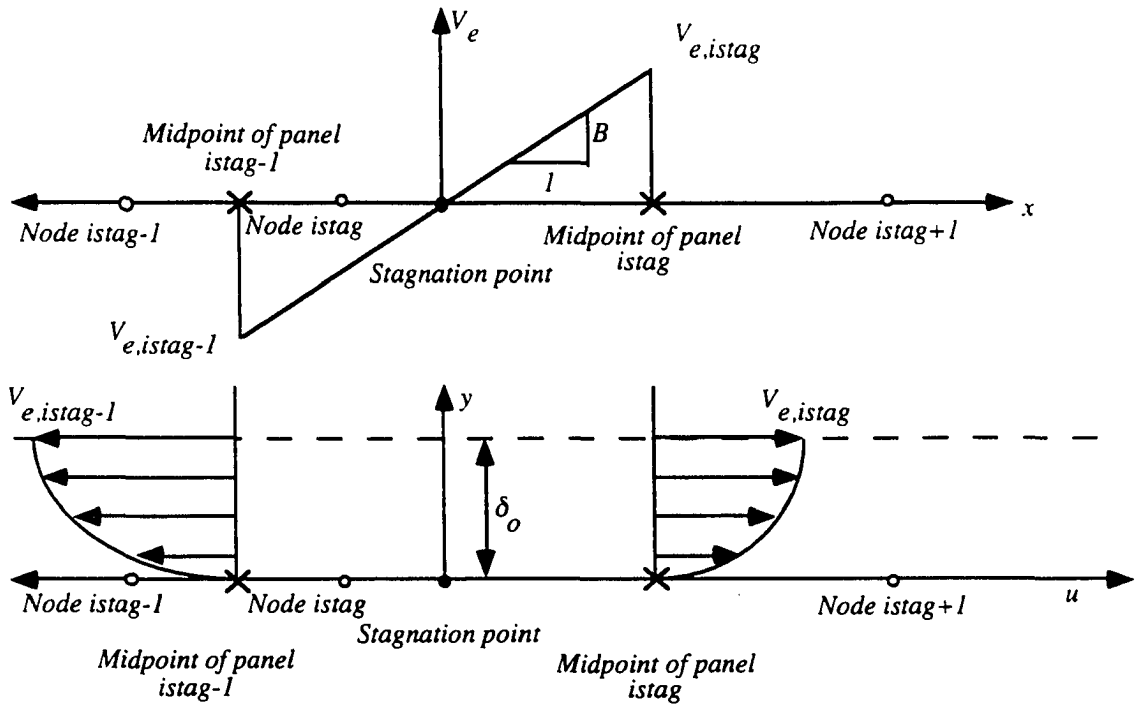


Figure 4.4: Stagnation point flow definitions

The stagnation point is determined by finding the place at which the tangential velocities first become positive. As one travels around the airfoil starting on panel "1" and moving to panel "2", the tangential velocities are negative (opposite to the sense of the direction of travel). See Chapter 3 for details of the panel numbering scheme. As one gets past the stagnation point, the tangential velocity becomes positive. Somewhere between the last negative velocity and the first positive velocity, the stagnation point exists. Consistent with the assumption that the tangential velocities vary linearly along the panel lengths, we assume that the distribution of tangential velocity is linear in the region of the stagnation point. The point at which the tangential velocity is zero can be found from a direct linear interpolation. The distances from the stagnation point to the first nodes on either side of the stagnation point are given by the panel geometry in that region. The goal is to find the stagnation velocity profile.

There are several factors within the boundary layer development at the stagnation point. Namely, for steady flow, there is a ratio, B, shown in Figure 4.4, which is the slope of the line that shows the edge velocity as a function of x. The usual procedure here is to allow a

Falkner-Skan solution at the stagnation point in which a similar solution is found for all points in the flow. Now, at the stagnation point, it is obvious that the flow has vertical components of velocity only. At the stagnation point, this is a point of symmetry. The observation is that the boundary layer thickness is constant in the region of the stagnation point, δ_o in Figure 4.4. In solving a similar solution, the ordinary differential equation is found to be

$$F''' + FF'' + 1 + (F')^2 = 0 \tag{4.33}$$

where F is given as

$$F(\eta) = \frac{-v}{\sqrt{Bv}} \tag{4.34}$$

(the vertical component of velocity, v, is in the numerator in Equation (4.34) while the kinematic viscosity, ν, is in the denominator) In this equation, η is a non-dimensional variable equal to:

$$\eta = y\sqrt{\frac{B}{\nu}} \tag{4.35}$$

so that the differentiation of F in Equation (4.33) is with respect to η. The horizontal component of velocity is related to the function, F, such that:

$$u = BxF'(\eta) \tag{4.36}$$

Since F is of third order in Equation (4.33), it requires three boundary conditions for a solution; namely

$$\begin{aligned} F(0) &= 0 \\ F'(0) &= 0 \\ F'(\infty) &= 1 \end{aligned} \tag{4.37}$$

The first of these two correspond no-slip and no penetration conditions at the surface and the third implies that the horizontal component of velocity approach Bx as η approaches infinity.

Equation (4.33) is non-linear in F and no known analytical solutions have ever been found. In that case, one resorts to a numerical approximation. A Runge-Kutta fourth order scheme was used to determine the distribution of F (and subsequently the vertical component of velocity distribution) at the stagnation point. The problem has boundary conditions at both ends of the domain ($0 \le \eta \le \infty$), but in order to use the Runge-Kutta scheme, initial conditions must be known for $F(0)$, $F'(0)$ and $F''(0)$. The first two are given and we must iterate on the other term. It turns out that the initial value of $F''(0)$ that matches the boundary condition at the other end of the domain is about 1.23259. Thus, with the correct value of $F''(0)$ known, we

can solve for the distribution of v. This provides us with the initial condition for the marching of the boundary layer solution. The boundary layer thickness in this problem is such that

$$\delta_o \cong 2.4\sqrt{\frac{\nu}{B}} \tag{4.38}$$

and outside the boundary layer, the vertical component of velocity varies such that the conservation of mass equation is satisfied, namely

$$v = -By \tag{4.39}$$

Ultimately, there are small modifications to this method when B changes in time. The initial condition for $F''(0)$ is not known and an iterative solution must be found. The results do not vary significantly from the steady case so although these modifications are made in the programming, they are not detailed here.

4.5 Numerical discretization

The objective is to find a method which approximates the partial differential equations (PDE) under consideration (Equations (4.18) and (4.19)). As mentioned in Chapter 3 of this dissertation, there are a variety of methods available to approximate this set of PDEs. Finite elements and finite differences both require a discretization process in that the values for the variables, u, v, and p are approximated at a discrete set of points. This set of points should be contained within the boundary layer. In the finite difference method that was chosen for this study, a set of grid points was chosen in the x-y coordinate system (see Figure 4.1) such that the origin of the coordinate system at any time is at the stagnation point (determined by the potential flow solution) and the x coordinate increases in the positive direction over the top of the airfoil to the trailing edge while the x coordinate decreases along the bottom of the airfoil to the trailing edge. All x coordinate points are set up to coincide with the midpoint of each of the panels (the control points for matching the no-penetration condition in the potential flow solution). The number of x coordinate points is equal to the number of panels plus one. The additional point is for the stagnation point.

Now to handle the coordinates perpendicular to the airfoil, a "region of interest" is determined. Within this region of interest, nodes for the discrete points are spaced such that there is a "stretching factor" involved. This allows the nodes to be spaced closer to the surface of the airfoil where the gradients are expected to be largest. This is realized in an uneven spacing. At any x coordinate, there may be a discretization in the y direction that looks similar to that shown in Figure 4.5. The ratio that is given to the program, β, is the ratio of one length,

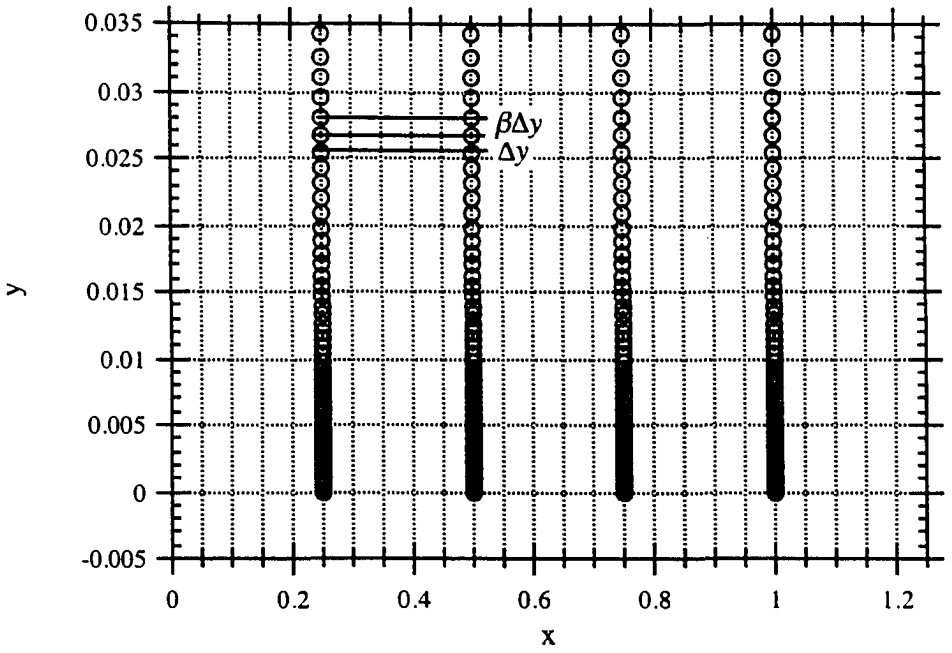


Figure 4.5: Typical boundary layer discretization

$\beta\Delta y$, to the previous, Δy as the y coordinate increases. Much work in looking at the effects of changing the β ratio found no significant differences in the results. The boundary layer thicknesses, displacement thicknesses, and momentum thicknesses as well as the distribution of shear stress were all similar as β was varied from 1.00 to up to 1.10. As such, the β ratio was set at 1.10 for all cases unless otherwise indicated.

Based on the solution process (described below), the boundary layer thickness is known throughout the flowfield. An initial guess is made at this "region of interest" and a grid of uniform height is developed where the spacing in the y direction is independent of the x coordinate (again, see Figure 4.5). Throughout the analysis, if the region of interest (shown as 0.035 in Figure 4.5), specified as y_f in the program, is less than 1.2 times the maximum boundary layer thickness, the region y_f is increased. Similarly, if y_f is found to be greater than 1.5 times the maximum boundary layer thickness (as the Reynolds number gets larger, the boundary layer thickness usually gets smaller), an adjustment is made. This means backing

up, putting the previous time step's solution into the adjusted discretized region though a linear interpolation of velocities in space and repeating the analysis for the current time step. The potential flow is not effected by this change. This monitoring is continued throughout the process. For future reference, it should be noted that the y coordinate is the same from one time step to the next (when no fluid is present), but that because the stagnation point can change with time, the x coordinate will change as well. A small change in the stagnation point from one time to the next will have to be accounted for in the finite difference formulation of the problem.

4.6 Finite difference formulas

After the discretization of the flowfield is complete via Section 4.5, we need to proceed to the solution. Repeated for clarity is the set of partial differential equations under consideration in this study- the conservation of momentum and the conservation of mass for a time-dependent, incompressible flow at high Reynolds numbers. If the flow has not made a transition to turbulence, the last term in Equation (4.18) is dropped from the analysis.

$$\rho\left(\frac{\partial \bar{u}}{\partial t}+\bar{u}\frac{\partial \bar{u}}{\partial x}+\bar{v}\frac{\partial \bar{u}}{\partial y}\right)=\rho\left(\frac{\partial V_e}{\partial t}+V_e\frac{\partial V_e}{\partial x}\right)+\frac{\partial}{\partial y}\left[\mu\left(\frac{\partial \bar{u}}{\partial y}\right)-\overline{\rho u'v'}\right] \tag{4.18}$$

$$\frac{\partial \bar{u}}{\partial x}+\frac{\partial \bar{v}}{\partial y}=0 \tag{4.19}$$

The objective in the finite difference formulation is to develop a set of equations that can be solved for information at the next time step. The premise of finite differences is quite simple. Suppose information were known about a variable, u, at time t_o. Then, suppose we wanted to estimate the value at time $t_0 + \Delta t$. Recalling the definition of a Taylor's series expansion, we might get the following:

$$u(t_o+\Delta t)=u(t_o)+\left.\frac{\partial u}{\partial t}\right|_{t=t_o}\Delta t+\left.\frac{\partial^2 u}{\partial t^2}\right|_{t=t_o}\frac{(\Delta t)^2}{2!}+\left.\frac{\partial^3 u}{\partial t^3}\right|_{t=t_o}\frac{(\Delta t)^3}{3!}+\ldots+\text{H.O.T.} \tag{4.40}$$

(H.O.T. stands for higher order terms) If we are willing to accept some error in our estimate, we can rewrite Equation (4.40) to solve for the first derivative with respect to time of u.

$$\left.\frac{\partial u}{\partial t}\right|_{t=t_o}=\frac{u(t_o+\Delta t)-u(t_o)}{\Delta t}-\left.\frac{\partial^2 u}{\partial t^2}\right|_{t=t_o}\frac{\Delta t}{2!}-\left.\frac{\partial^3 u}{\partial t^3}\right|_{t=t_o}\frac{(\Delta t)^2}{3!}+\ldots+\text{H.O.T.} \tag{4.41}$$

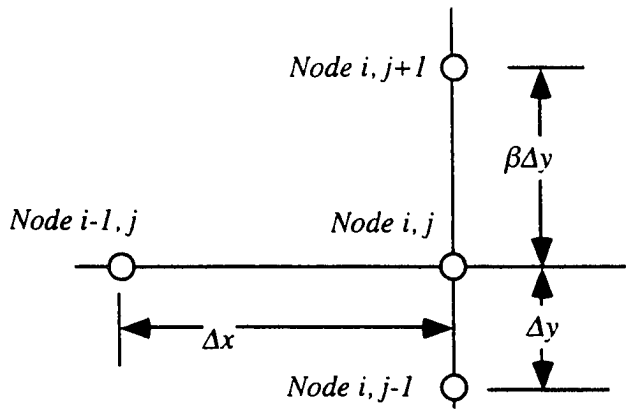


Figure 4.6: Computational cell for the gas-dynamic boundary layer

If we retain only the first term on the right hand side of Equation (4.41) as our approximation to the first derivative with respect to time, we say that the approximation has a truncation error of *first order*. As the time increment, Δt, gets smaller, the approximation will get better in a linear fashion. There are second order approximations that will get accurate more quickly, (i.e. the truncation error is of second order), but these involve more discrete points at which information must be known or solved.

Let us look at a short naming convention for the discretization that will be used consistently throughout this analysis. Take a variable, u, which is a function of time and two space coordinates, x and y. Our discrete value of the estimate of u might look like "$u^k_{i,j}$". We will use the subscript i to delineate the x coordinate (goes from 1 to NN+1) and the index j corresponds to the y coordinate (goes from 1 at the bottom to NJ at the outer edge of the region of interest). That leaves k as the time step marker. This is the convention used in this study.

Let us write an approximation to each of the terms in Equation (4.18). Assume that the flow is in the direction of increasing i, i.e. on the top of the airfoil. The bottom of the airfoil is handled in a similar manner, except that we search for a solution at station i where information is known at station $i+1$ (the subscript decreases from *istag* to *1*). The computational cell that we are using for flow on top of the airfoil is shown in Figure 4.6.

This is a first order approximation to the first derivative with respect to time of the u velocity.

$$\frac{\partial \bar{u}}{\partial t} = \frac{u_{i,j}^{k+1} - u_{i,j}^{k}}{\Delta t} \tag{4.42}$$

The next term is approximated as:

$$\bar{u}\frac{\partial \bar{u}}{\partial x} = u_{i-1,j}^{k+1}\left(\frac{u_{i,j}^{k+1} - u_{i-1,j}^{k+1}}{\Delta x}\right) \tag{4.43}$$

This term is a little different. We would normally write the equation as

$$\bar{u}\frac{\partial \bar{u}}{\partial x} = u_{i,j}^{k+1}\left(\frac{u_{i,j}^{k+1} - u_{i-1,j}^{k+1}}{\Delta x}\right) \tag{4.44}$$

but this would give us a non-linear equation for $u_{i,j}^{k+1}$. The process used in this study is to find the solution for the u velocities at station i and move on to the next station, i+1. Information at station i-1 is known as we search for the solution at station i. The process used here is known as "linearizing by lagging". It allows for a linearized set of equations to be solved at each x station. As you can see with the next approximation,

$$\bar{v}\frac{\partial \bar{u}}{\partial y} = v_{i-1,j}^{k+1}\left(\frac{u_{i,j+1}^{k+1} + (\beta^2 - 1)u_{i,j}^{k+1} - \beta^2 u_{i,j-1}^{k+1}}{\beta(\beta+1)\Delta y}\right) \tag{4.45}$$

the values for $u_{i,j+1}^{k+1}, u_{i,j}^{k+1}$ and $u_{i,j-1}^{k+1}$ are unknown at each station, i. These unknowns show up in the next approximation as well.

$$\frac{\partial}{\partial y}\left[\mu\left(\frac{\partial u}{\partial y}\right)\right] = \mu\left(\frac{\partial^2 u}{\partial y^2}\right) = \mu\frac{2}{\beta(1+\beta)(\Delta y)^2}\left(u_{i,j+1}^{k+1} - (1+\beta)u_{i,j}^{k+1} - \beta u_{i,j-1}^{k+1}\right) \tag{4.46}$$

The approximations to the pressure gradient are handled as one might expect.

$$\frac{\partial V_e}{\partial t} = \frac{V_{e,i}^{k+1} - V_{e,i}^{k}}{\Delta t} \tag{4.47}$$

$$V_e\frac{\partial V_e}{\partial x} = V_{e,i}^{k+1}\left(\frac{V_{e,i}^{k+1} - V_{e,i-1}^{k+1}}{\Delta x}\right) \tag{4.48}$$

For each station, i, at which we wish to find the solution for $u_{i,j}^{k+1}$ we write a series of equations where the j subscript ranges from 2 to *NJ-1*. The set of equations is written as

$$B_j u_{i,j-1}^{k+1} + D_j u_{i,j}^{k+1} + A_j u_{i,j+1}^{k+1} = C_j \tag{4.49}$$

where each of the approximations has been multiplied by Δt and divided by ρ. We get:

$$B_j = -\frac{\Delta t v_{i-1,j}^{k+1}\beta}{(\beta+1)\Delta y} - \frac{2\nu\Delta t}{(\Delta y)^2(1+\beta)} \tag{4.50}$$

$$D_j = 1.0 + \frac{\Delta t}{\Delta x}u_{i-1,j}^{k+1} + \frac{\Delta t v_{i-1,j}^{k+1}(\beta^2-1)}{\beta(\beta+1)\Delta y} + \frac{2\nu\Delta t}{(\Delta y)^2\beta} \tag{4.51}$$

$$A_j = \frac{\Delta t v_{i-1,j}^{k+1}\beta}{(\beta+1)\Delta y} - \frac{2\nu\Delta t}{(\Delta y)^2(1+\beta)\beta} \tag{4.52}$$

and

$$C_j = u_{i,j}^k + \frac{\Delta t}{\Delta x}\left(u_{i-1,j}^{k+1}\right)^2 + V_{e,i}^{k+1} - V_{e,i-1}^{k+1} + \frac{\Delta t}{\Delta x}V_{e,i}^{k+1}\left(V_{e,i}^{k+1} - V_{e,i-1}^{k+1}\right) \tag{4.53}$$

This linearized set of equations is solved in an *implicit* manner. As the development of this project progressed, it was clear that numerical instability should be eliminated at every opportunity. This is the main feature of an implicit method. The trade-off is that a set of equations must be solved at each station rather than a simple prediction of each point based on known information. This is the *explicit* method and can be numerically unstable. Small errors in the solution can grow if the time-steps are not small enough relative to other parameters. Numerical stability, therefore, is not an issue in this portion of the study due to the implicit method of solution.

Now, the set of equations given by Equation (4.49) needs to be solved at each station before we can move on. The assumption, for the moment, is that the flow is laminar and the inclusion of the last term in Equation (4.18) is not needed yet. If we want to include the turbulence, we could estimate the turbulent shear stress term as

$$\begin{aligned}\frac{\partial}{\partial y}\left[-\rho\overline{u'v'}\right] &= \frac{\partial}{\partial y}\left[\mu_T\frac{\partial\bar{u}}{\partial y}\right] \\ &= \frac{\partial}{\partial y}\left(\rho l_m^2\left|\frac{\partial\bar{u}}{\partial y}\right|\frac{\partial\bar{u}}{\partial y}\right) \\ &= \frac{2}{(1+\beta)\Delta y}\left[\left(\mu_T\right)_{i-1,j+1/2}^{k+1}\left(\frac{u_{i,j+1}^{k+1}-u_{i,j}^{k+1}}{\beta\Delta y}\right) - \left(\mu_T\right)_{i-1,j-1/2}^{k+1}\left(\frac{u_{i,j}^{k+1}-u_{i,j-1}^{k+1}}{\Delta y}\right)\right]\end{aligned} \tag{4.54}$$

where

$$\left(\mu_T\right)_{i-1,j+1/2}^{k+1} = \rho\left(\frac{\left(l_{m,i-1,j}^{k+1}\right)^2 + \left(l_{m,i-1,j+1}^{k+1}\right)^2}{2}\right)\left|\frac{u_{i-1,j+1}^{k+1} - u_{i-1,j}^{k+1}}{\beta\Delta y}\right| \tag{4.55}$$

$$\left(\mu_T\right)_{i-1,j-1/2}^{k+1} = \rho\left(\frac{\left(l_{m,i-1,j}^{k+1}\right)^2 + \left(l_{m,i-1,j-1}^{k+1}\right)^2}{2}\right)\left|\frac{u_{i-1,j}^{k+1} - u_{i-1,j-1}^{k+1}}{\Delta y}\right|$$

This formulation is suggested by Anderson, Tannehill and Pletcher [1]. Just as the previous situation, if one were interested in making a more accurate model of the mixing length, one would base the information on information at the current x coordinate, but in this formulation, the modeling of the mixing length is based on information at the previous i index, in essence, a lagging. Otherwise, there would exist the need to iterate on a solution for the velocity profile within the boundary layer and recalculate the mixing lengths.

This term will modify the factors found in Equations (4.50),(4.51), and (4.52). Namely the new terms are changed such that:

$$B_j\big|_{modified} = B_j\big|_{old} - \frac{\Delta t}{(1+\beta)(\Delta y)^2}\left\{\left(l_{m,i-1,j}^{k+1}\right)^2 + \left(l_{m,i-1,j+1}^{k+1}\right)^2\right\}\left|\frac{u_{i-1,j+1}^{k+1} - u_{i-1,j}^{k+1}}{\beta\Delta y}\right| \tag{4.56}$$

$$\begin{aligned} D_j\big|_{modified} = D_j\big|_{old} &+ \frac{\Delta t}{\beta(1+\beta)(\Delta y)^2}\left\{\left\{\left(l_{m,i-1,j}^{k+1}\right)^2 + \left(l_{m,i-1,j+1}^{k+1}\right)^2\right\}\left|\frac{u_{i-1,j+1}^{k+1} - u_{i-1,j}^{k+1}}{\beta\Delta y}\right|\right\} \\ &+ \frac{\Delta t}{(1+\beta)(\Delta y)^2}\left\{\left(l_{m,i-1,j}^{k+1}\right)^2 + \left(l_{m,i-1,j+1}^{k+1}\right)^2\right\}\left|\frac{u_{i-1,j+1}^{k+1} - u_{i-1,j}^{k+1}}{\beta\Delta y}\right| \end{aligned} \tag{4.57}$$

$$A_j\big|_{modified} = A_j\big|_{old} - \frac{\Delta t}{\beta(1+\beta)(\Delta y)^2}\left\{\left(l_{m,i-1,j}^{k+1}\right)^2 + \left(l_{m,i-1,j-1}^{k+1}\right)^2\right\}\left|\frac{u_{i-1,j}^{k+1} - u_{i-1,j-1}^{k+1}}{\Delta y}\right| \tag{4.58}$$

There are two modifications that are needed based on the boundary conditions. It is assumed that at the edge of the region of interest, the u component of velocity must match the freestream velocity at that station. Therefore, C_{NJ-1} is modified such that:

$$C_{NJ-1}\big|_{modifified} = C_{NJ-1}\big|_{old} - V_{e,i}^{k+1} A_{NJ-1} \tag{4.59}$$

Similarly, there is a known u velocity at the bottom of the boundary layer due to the no-slip condition. When deicing fluid is not present, this value is zero, but if there is a finite slip velocity (as there generally is with the fluid present, C_2 is modified in the following way

$$C_2\big|_{modifified} = C_2\big|_{old} - u_{i,1}^{k+1} B_2 \tag{4.60}$$

We now have a complete set of equations modified by the boundary conditions for the solution of the u component of velocity at station i based on known information from the potential flow, boundary conditions of no-slip, and information from the previous station. The form of Equation (4.49) is such that it is considered to be tridiagonal. An algorithm known as the Thomas algorithm takes advantage of this fact and the solution at each station is reached quickly.

Next in the process is to solve for the v component of velocity. This is based on the continuity equation (Equation (4.19)). The v component of velocity is estimated at each node by:

$$v_{i,j}^{k+1} = v_{i-1,j}^{k+1} - \frac{\Delta y}{\Delta x}\left(u_{i,j}^{k+1} - u_{i-1,j}^{k+1} + u_{i,j-1}^{k+1} - u_{i-1,j-1}^{k+1}\right) \tag{4.61}$$

Once the velocity distribution is known, there are several parameters that are calculated. The boundary layer thickness is found, and the displacement thickness and momentum thickness are calculated using a Simpson's Rule estimate of the integrals in Equations (4.28) and (4.29). With the displacement thickness and momentum thickness available, the shape factor is calculated as the ratio of the two and the Reynolds number based on displacement thickness is calculated. Should the Reynolds number based on displacement thickness exceed the critical value from the curve fit given in Equation (4.32), the flow is assumed to be turbulent at the next station and the terms relevant to the Reynolds shear stress are added to the solution of the u component of velocity process.

Finally, the shear stress at the bottom of the boundary layer is calculated. Even in turbulent flow, it is assumed the shear stress is proportional to the velocity gradient and the dynamic viscosity of the air. Recall that the lowest regions of the turbulent boundary layer are considered to be "laminar". This means that the shear stress in the fluid is linearly proportional to these two quantities. The shear stress at the wall is estimated using the first four data points in the boundary layer to be (if Δy is $y_{i,2}$-$y_{i,1}$ and β is constant):

$$\begin{aligned}\tau_w = &-u_{i,1}^{k+1}\left(\frac{(3+4\beta+3\beta^2+3\beta^3)}{\Delta y(1+2\beta+2\beta^2+\beta^3)}\right) + u_{i,2}^{k+1}\left(\frac{(1+\beta+\beta^2+3\beta^3)}{\Delta y(\beta^2)}\right) \\ &-u_{i,3}^{k+1}\left(\frac{(1+\beta+\beta^2)}{\Delta y(\beta^3+\beta^4)}\right) + u_{i,4}^{k+1}\left(\frac{1}{\Delta y(1+\beta+\beta^2)\beta^3}\right)\end{aligned} \tag{4.62}$$

Notice the inclusion of the term that goes with $u_{i,1}^{k+1}$. This term will be zero when the fluid is not present.

Finally, with all of these parameters known, the boundary layer development is solved for each time step, first on the top of the airfoil and then on the bottom.

4.7 Integrating the gas-dynamic boundary layer with the potential flow

The main factors in integrating the potential flow results with the gas-dynamic boundary layer module of the programming are 1) realizing the potential flow effects the gas-dynamic boundary in the boundary condition applied to the solution of the u component of velocity, 2) the stagnation point due to the potential flow solution changes with time, changing the coordinates in the x direction with time and 3) there is a viscid-inviscid interaction in which the gas-dynamic boundary layer's displacement thickness will effect the potential flow result.

Since the stagnation point is the origin of the gas-dynamic boundary layer coordinate system and the stagnation point can change with time, we should account for this in the formulation of the boundary layer equations. Suppose we have a coordinate system, (x, y), at time $t = t_o$ and an origin at $x = 0$. Then, upon a new potential flow solution, the stagnation point (and the origin of the gas-dynamic boundary layer) moves at a time, $t = t_o + \Delta t$. Suppose there is a coordinate transformation such that

$$\begin{aligned} x &= x(\zeta,\tau) \\ y &= y(\eta,\tau) \\ t &= \tau \end{aligned} \tag{4.63}$$

where ζ and η are the current normal and tangential coordinates in the computational domain. We must make an adjustment to account for the motion of this coordinate system. The boundary layer equations are valid in normal and tangential coordinates, but if the coordinates ζ and η depend on time, ($\zeta = \zeta(x, t)$ and $\eta = \eta(y, t)$) , then

$$\begin{aligned} \frac{\partial}{\partial x} &= \underbrace{\frac{\partial}{\partial \zeta}\frac{\partial \zeta}{\partial x}}_{\frac{\partial}{\partial \zeta}1} + \underbrace{\frac{\partial}{\partial \eta}\frac{\partial \eta}{\partial x}}_{\frac{\partial}{\partial \eta}0} + \underbrace{\frac{\partial}{\partial \tau}\frac{\partial \tau}{\partial x}}_{\frac{\partial}{\partial \tau}0} \\ &= \frac{\partial}{\partial \zeta} \end{aligned} \tag{4.64}$$

Similarly,

$$\frac{\partial}{\partial y} = \underbrace{\frac{\partial}{\partial \zeta}\frac{\partial \zeta}{\partial y}}_{\frac{\partial}{\partial \zeta}0} + \underbrace{\frac{\partial}{\partial \eta}\frac{\partial \eta}{\partial y}}_{\frac{\partial}{\partial \eta}1} + \underbrace{\frac{\partial}{\partial \tau}\frac{\partial \tau}{\partial y}}_{\frac{\partial}{\partial \tau}0} \tag{4.65}$$

$$= \frac{\partial}{\partial \eta}$$

and

$$\frac{\partial}{\partial t} = \underbrace{\frac{\partial}{\partial \zeta}\frac{\partial \zeta}{\partial t}}_{\frac{\partial}{\partial \zeta}\frac{\partial \zeta}{\partial t}} + \underbrace{\frac{\partial}{\partial \eta}\frac{\partial \eta}{\partial t}}_{\frac{\partial}{\partial \eta}\frac{\partial \eta}{\partial t}} + \underbrace{\frac{\partial}{\partial \tau}\frac{\partial \tau}{\partial t}}_{\frac{\partial}{\partial \tau}1} \tag{4.66}$$

$$= \frac{\partial}{\partial \zeta}\frac{\partial \zeta}{\partial t} + \frac{\partial}{\partial \eta}\frac{\partial \eta}{\partial t} + \frac{\partial}{\partial \tau}$$

So, in the computational domain, (ζ, η, τ), we must include the time derivatives of the two space coordinates in the formulation of the first derivatives of any of the variables with respect to time. This means, for example, that

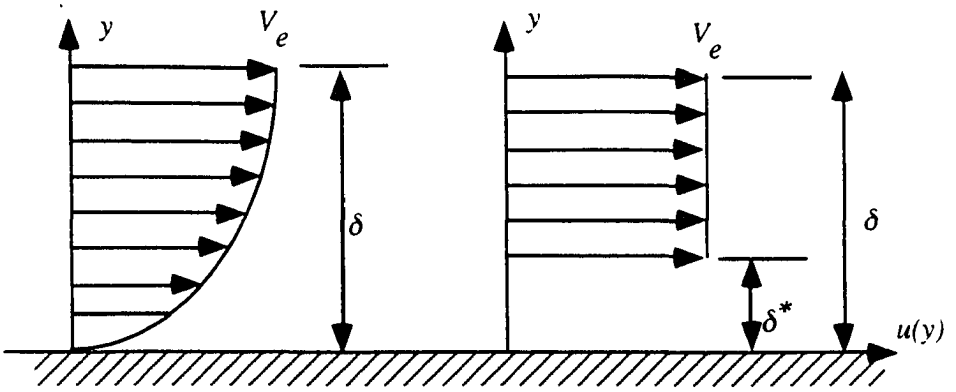
$$\frac{\partial u}{\partial t} = \frac{\partial u}{\partial \tau} + \frac{\partial \zeta}{\partial t}\frac{\partial u}{\partial \zeta} + \frac{\partial \eta}{\partial t}\frac{\partial u}{\partial \eta} \tag{4.67}$$

and the conservation of momentum equation (Equation (4.18)) is rewritten in terms of computation coordinates as:

$$\rho\left(\frac{\partial \bar{u}}{\partial \tau} + \left(\bar{u} + \frac{\partial \zeta}{\partial t}\right)\frac{\partial \bar{u}}{\partial \zeta} + \left(\bar{v} + \frac{\partial \eta}{\partial t}\right)\frac{\partial \bar{u}}{\partial \eta}\right) = \rho\left(\frac{\partial V_e}{\partial \tau} + \left(V_e + \frac{\partial \zeta}{\partial t}\right)\frac{\partial V_e}{\partial \zeta}\right) + \frac{\partial}{\partial \eta}\left[\mu\left(\frac{\partial u}{\partial \eta}\right) - \overline{\rho u'v'}\right] \tag{4.68}$$

For the conservation of mass equation, no adjustments are needed. These correction terms are included in the results of the gas-dynamic boundary layer. The $\frac{\partial \zeta}{\partial t}$ term will come from a change in the stagnation point, and eventually, the $\frac{\partial \eta}{\partial t}$ term will come from the changing of the grid with time due to the height of the deicing fluid at a given position along the airfoil changing with time.

Finally, in integrating the gas-dynamic boundary layer results with the potential flow, there is the issue of viscid-inviscid interaction. The displacement thickness that develops on an airfoil will have an effect on the potential flow solution. Physically, the displacement thickness



Fully developed boundary layer *Equvilalent mass flow*

Figure 4.7: Concept of a displacement thickness

produces an effect such that it thickens the airfoil by the amount δ^*. The same amount of mass flow exists at a given point if the fully-developed boundary layer is present or a flow which has a uniform distribution of velocity exists though a height of δ- δ^* (See Figure 4.7).

In accounting for the displacement thickness, Moran [29] suggests that a modification is needed to the no-penetration condition in the potential flow analysis. A normal velocity is prescribed at the midpoint of each panel such that:

$$V_n = \frac{\partial}{\partial x}\left(V_e \delta^*\right) \tag{4.69}$$

The process is an iterative one in which the flow is analyzed as if inviscid ($V_n = 0$) and the displacement thickness is found from the solution of the boundary layer and the input tangential velocities, then the condition in Equation (4.69) is used to find the corrected normal velocities. The normal velocities on each panel are implemented by modifying the b vector in the potential flow analysis similar to the method used to account for trailing vortices. Once the normal velocities are found, a new potential flow solution is calculated and the process continues until the normal velocities do not differ from one iteration to the next. The overall effect of including the displacement thickness is to slightly lower the section lift coefficients and to slightly increase the section drag coefficients. We do this in anticipation of including the deicing fluid as a displacement thickness effect.

4.8 Typical gas-dynamic boundary layer results

With the process described above, we will now look at some of the results from our example problem prior to a full analysis in the RESULTS chapter of this dissertation.

Take the NACA airfoil initially at 1° angle of attack used in the examples of Chapter 3. It is well-known that the boundary layer development has a dependence on Reynolds number. Before any kind of boundary layer analysis on the accelerating flow can be completed, as mentioned in the take-off analysis simulation of the problem in Chapter 3, there is a steady-state of potential flow that is developed on the airfoil. For the start-up speed of 3.5 m/s on the 1.0 m long airfoil and assuming standard temperature and pressure for the air, prior to the acceleration, we arrive at a steady-state distribution of pressure that includes the effects of the displacement thickness that looks like Figure 4.8 on the top surface. The Reynolds number based on chord length in this example problem is $2.40(10^5)$. The bottom surface is much like the top; the transition point is different, but the shapes of the curves are similar. We will concentrate on the top of the airfoil for the moment.

The boundary layer thickness for the example solution at a Reynolds number of $2.40(10^5)$ appears in Figure 4.9 along with distributions of displacement thickness and momentum thickness. The distances shown in the abscissa in Figure 4.9 are nondimensional (y/c). One observation is that there is some "jumpiness" in the result of the values for the displacement thickness. This is to be expected as merely discretization error. Throughout the analysis, 125 nodes in the y direction were used and unless otherwise stated, 125 nodes were used to discretize the region of interest that surrounds the boundary layer. In this example problem as well as all other work, the coordinate x in the graphs that display boundary layer parameters is known to be the (non-dimensional) distance (since the chord length is always set to unity) along the surface of the airfoil from the stagnation point.

Figure 4.10 shows the shape factor distribution for this steady flow. Remember, the effect of displacement thickness on the potential flow has been incorporated into all these data. From Figure 4.10 it is clear that there is a transition made to turbulent flow at approximately 25% of the chord length. The shape factor changes from a value near 2.6 for the laminar flow to about 1.7 for the turbulent flow. As the boundary layer nears separation (the shear stress on the airfoil approaches zero), the shape factor increases. This is indicative of the nearly separated turbulent flow.

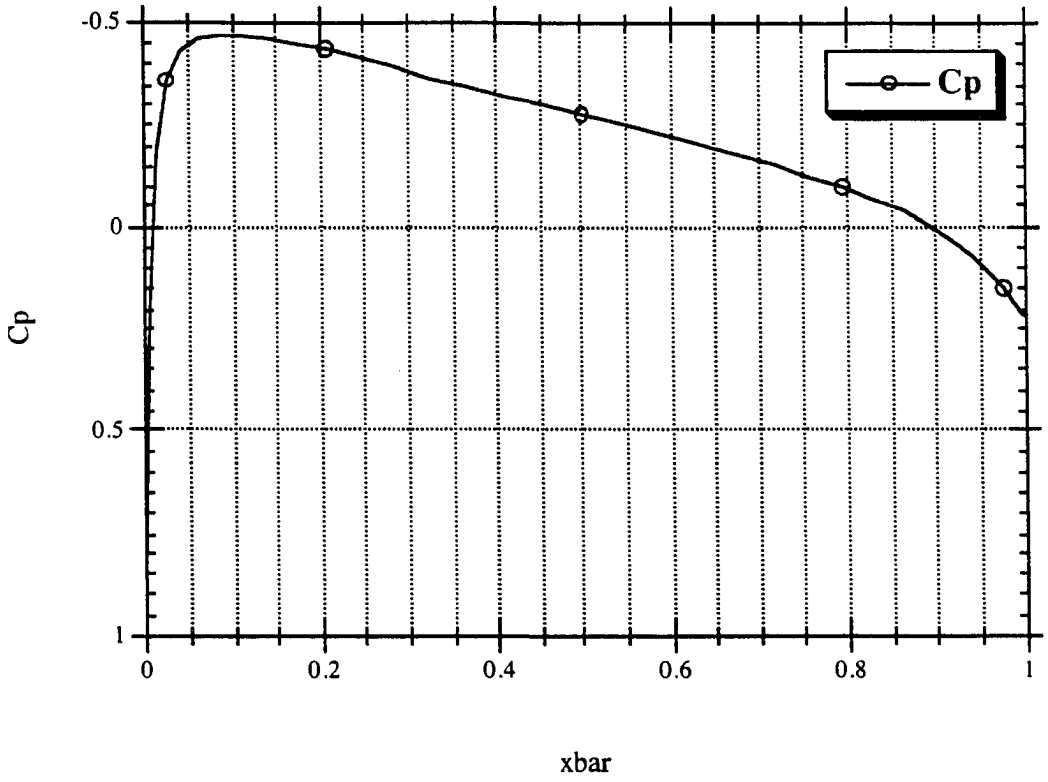


Figure 4.8: Pressure distribution on top of the NACA 0012 airfoil at 1° AOA and Re=2.40(10^5)

Finally, a distribution of shear stress in the form of a skin friction coefficient is shown in Figure 4.11. The skin friction coefficient at the wall is defined to be:

$$c_f = \frac{\tau_w}{\frac{1}{2}\rho V_e^2} \tag{4.70}$$

In Figure 4.11 it is even more obvious where the transition point exists. For this particular steady-flow problem, the section lift coefficient was found to be 0.1134. This compares to the potential flow prediction of 0.1204. As mentioned previously, the effect of

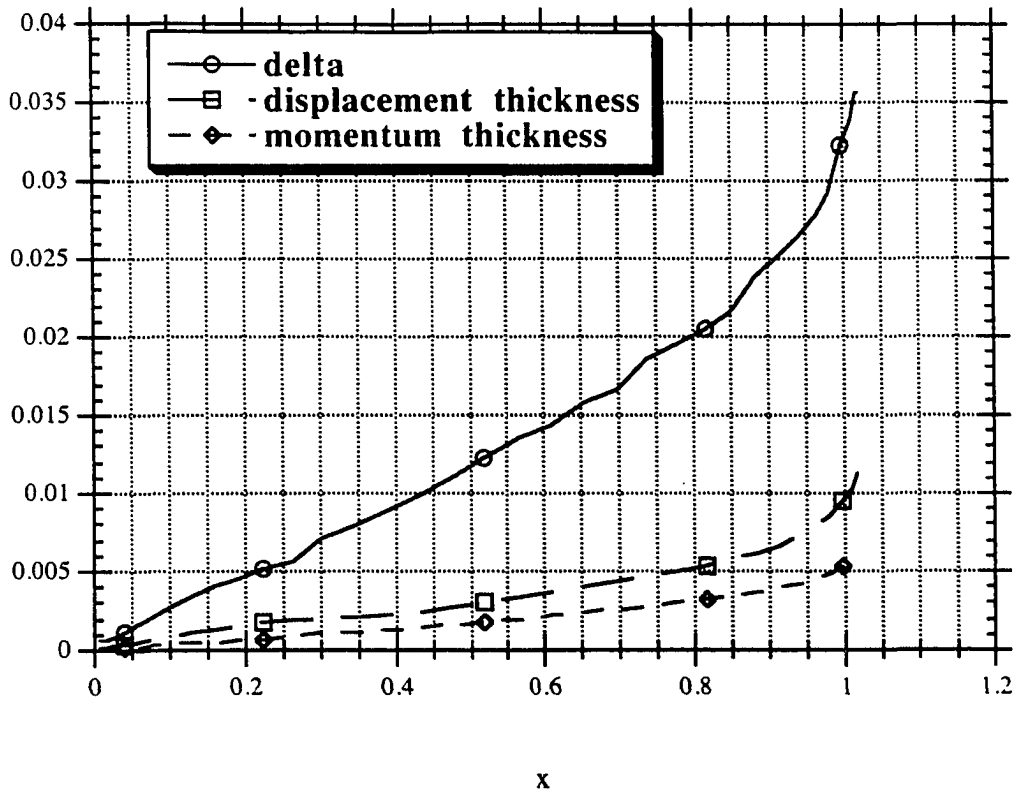


Figure 4.9: Boundary layer parameters for the example problem

correcting for the displacement thickness is to slightly lower the section lift coefficient. This change is more pronounced at higher angles of attack.

Within the flow, one might be interested in the velocity profiles predicted. Prior to transition, a laminar velocity profile is exhibited. Figure 4.12 shows the ratio of vertical distance to the boundary layer thickness (y/δ) as a function of the non-dimensional velocity, u/V_e. Also shown in Figure 4.12 is the velocity profile at about 75% of chord. Here the flow is clearly turbulent and the velocities should be thought of as time-averaged velocities.

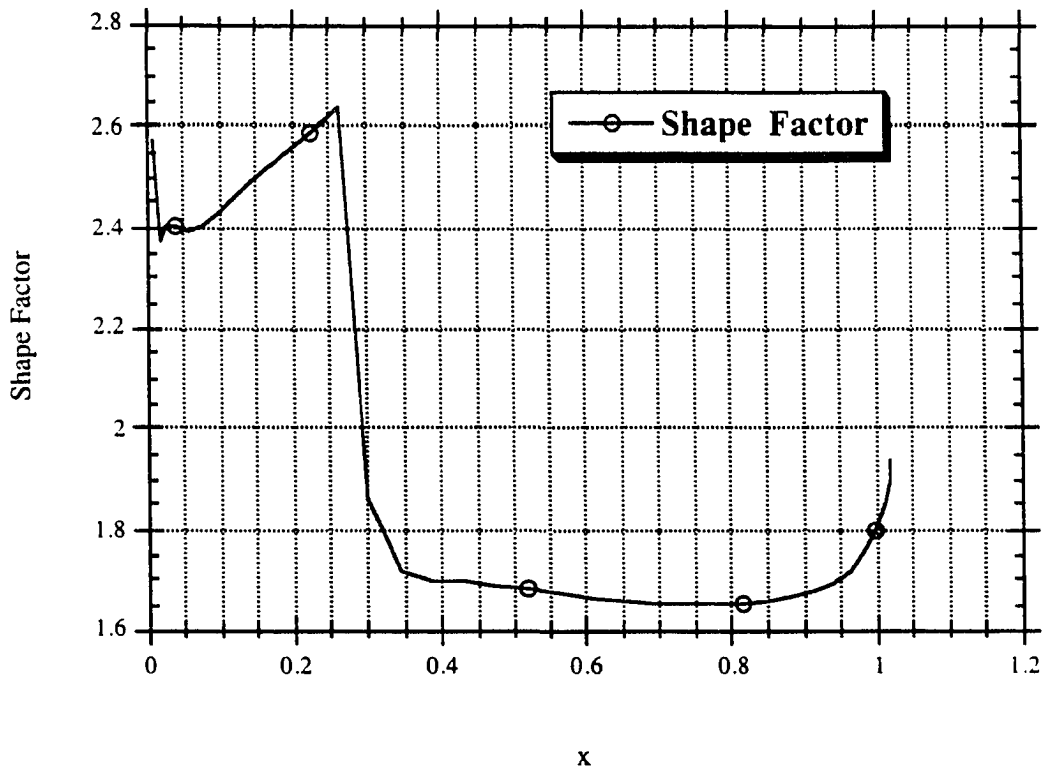


Figure 4.10: Example problem distribution of shape factor

Another way to look at the velocity profiles (particularly turbulent velocity profiles) is to plot the non-dimensional velocity u^+ as a function of the non-dimensional length y^+ (See Equations (4.21) and (4.22)).

Figure 4.13 shows the velocity profile at stations near the 75% chord mark and near the trailing edge (100% of chord). As the skin friction (shear stress at the wall) decreases as one moves further along the chord from 75% to the trailing edge, the value for the friction velocity decreases and the value for the non-dimensional velocity u^+ increases. Since the mixing length model in the inner layer is based on y^+, which is linearly proportional to the friction velocity, it should be noted that one of the shortcomings of the mixing length method of turbulence modeling is the relatively poor prediction of turbulent stresses near separation.

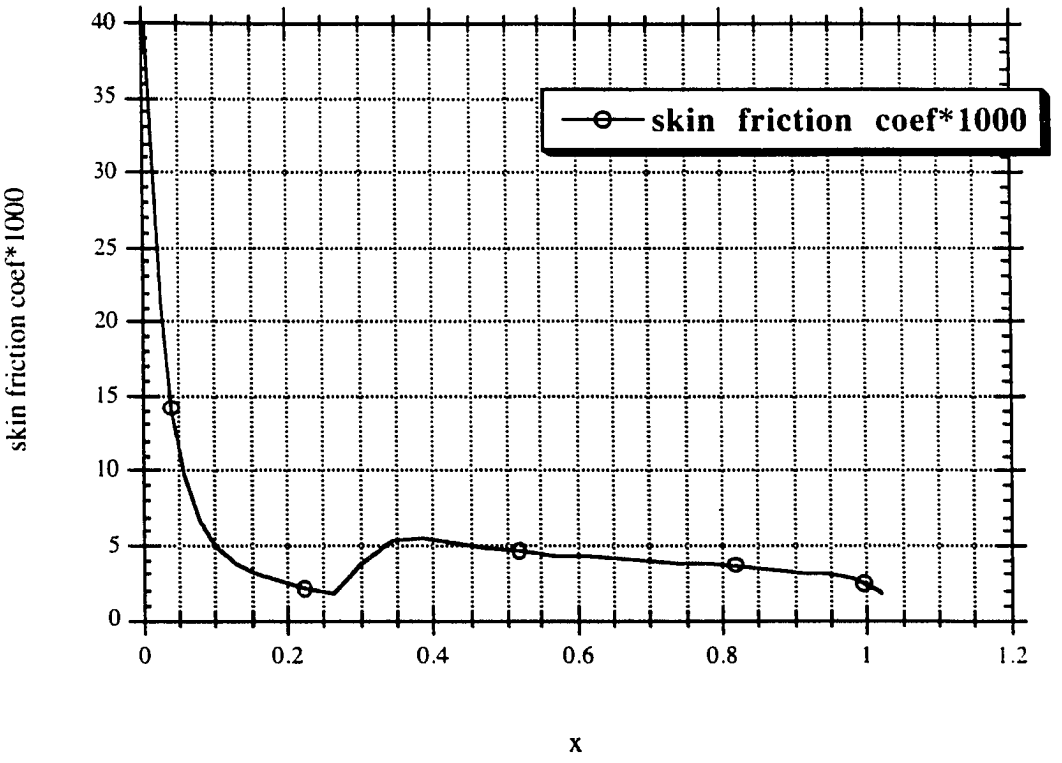


Figure 4.11: Skin friction as a function of position for the example problem

For the example problem, the section lift and drag coefficients as a function of time are plotted in Figure 4.14.

In the example problem, the freestream velocity begins at 3.5 m/s and increases at a rate of 2.5 m/s^2 until the rotation speed of 41.0 m/s. In this acceleration phase, the initial angle of attack is 1.0 degrees and at rotation, increases at 3.5 degrees per second as the freestream velocity is allowed to continue its acceleration. During this period, the acceleration causes the transition point to initially move to the back and then work its way back to the front. The acceleration is "helpful" to the boundary layer (at the low angle of attack in the initial take-off roll) in that it is deemed a favorable pressure gradient in the time dependent Euler's Equation approximation to the pressure gradient. As this effect becomes less important (at higher

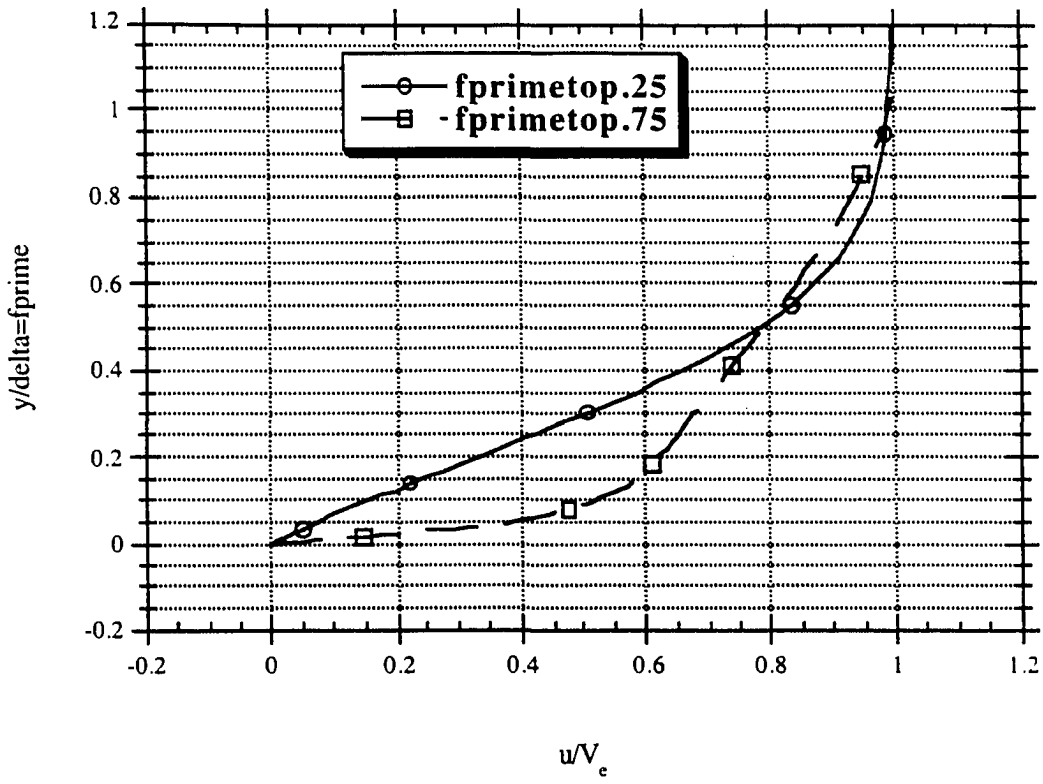


Figure 4.12: Typical velocity profiles for the example problem at 25% and 75% of chord

speeds), the transition point moves toward the leading edge because of increasing Reynolds number. During rotation, the transition point on the top of the airfoil moves even further to the front of the airfoil as the minimum pressure moves ahead as well.

In Figure 4.14 the maximum lift coefficient is found to be 1.5776 at angle of attack of 15.719 degrees. This is determined as the point at which the flow separates near the trailing edge. The boundary layer equations are ill-equipped to handle separated flow and in any case, once the separation is predicted, it is assumed that the performance will suffer. No attempt is made to go beyond this point and in fact, as the deicing fluid is included in the analysis, we will attempt to show the effect of the fluid as a loss in maximum lift coefficient.

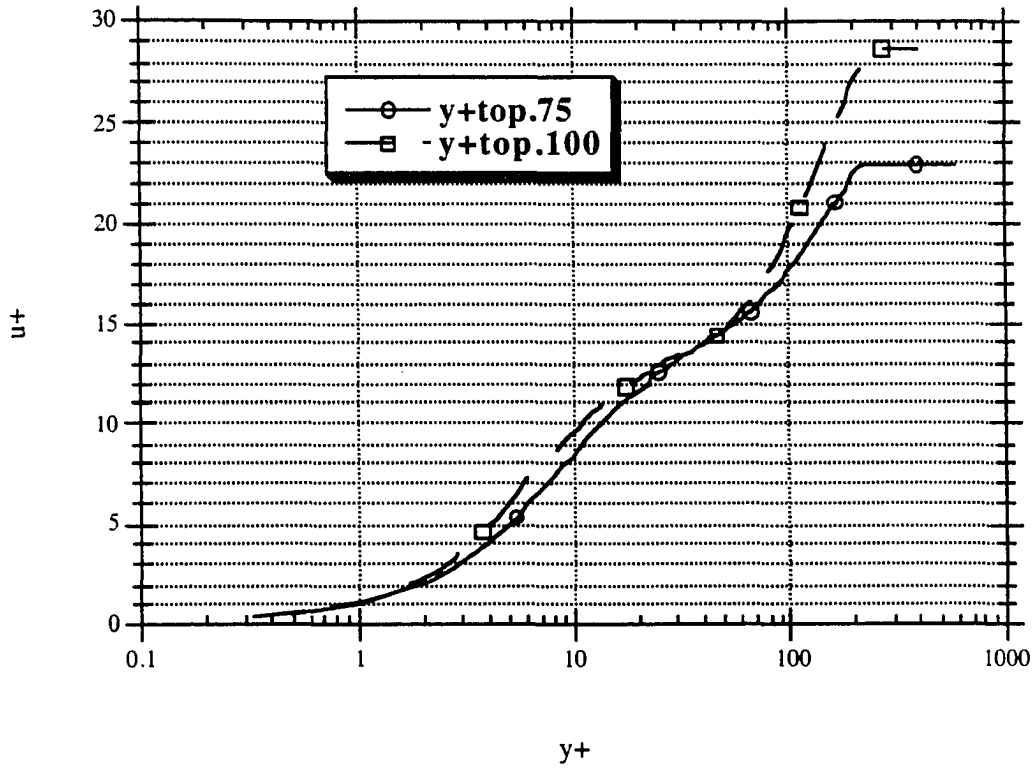


Figure 4.13: Example turbulent velocity profiles at 75% and 100% of chord

For comparison purposes, it may be useful to look at the differences in the predicted lift coefficients when the gas-dynamic boundary layer is included and when it is excluded from the analysis. Shown in Figure 4.15 are the predicted values for the section lift coefficient as a function of angle of attack for the same rotation rate, initial angle of attack and freestream velocities and acceleration rates used in the example problem.

As expected, the effect of including the displacement thickness reduces the section lift coefficient. The drag coefficients increase with the inclusion of the displacement thickness as well.

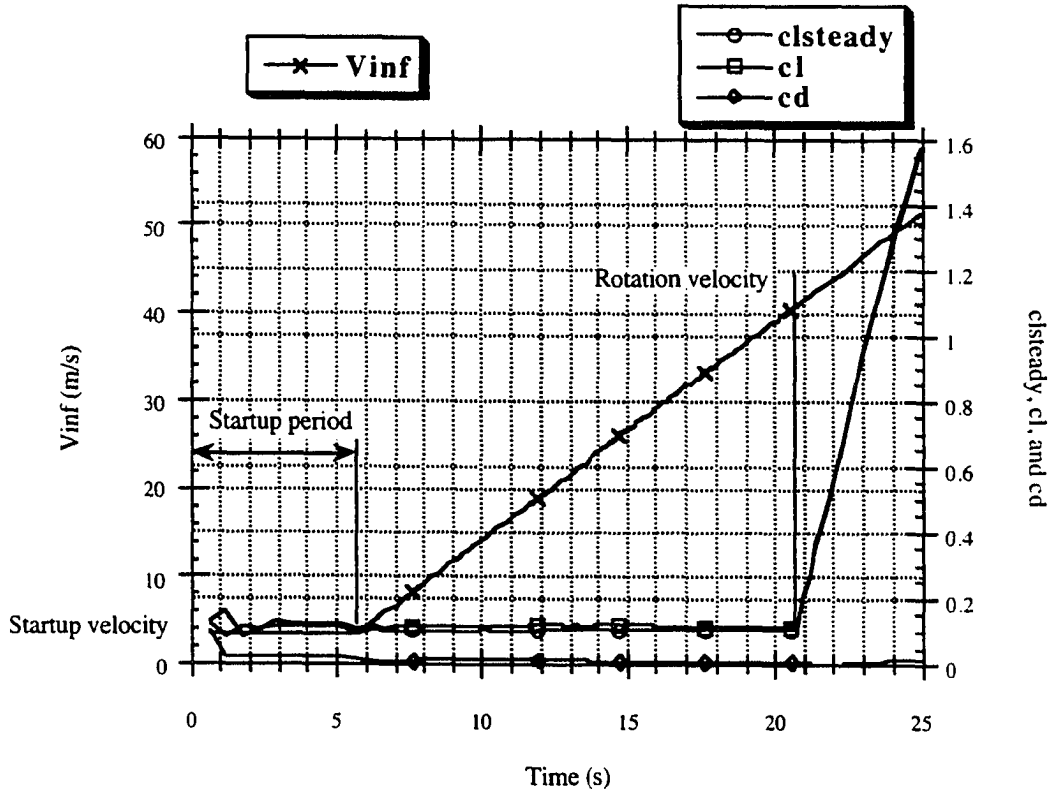


Figure 4.14: Section lift and drag coefficients when the gas-dynamic boundary layer is included with the potential flow analysis

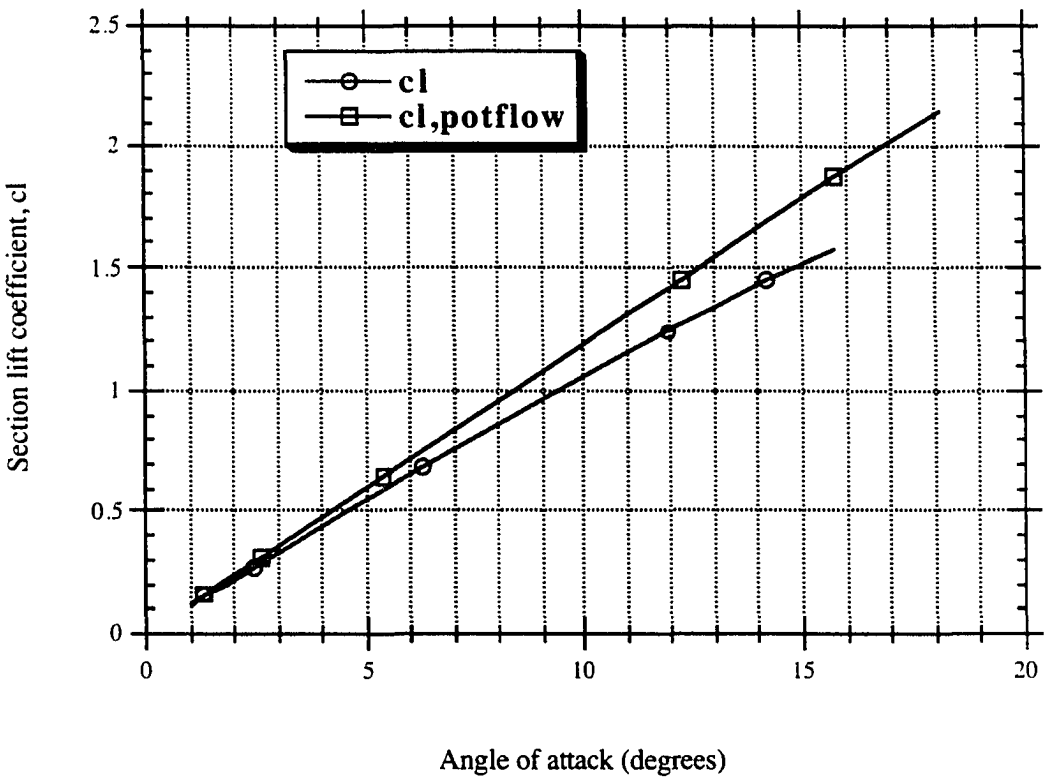


Figure 4.15: Section lift coefficients as a function of angle of attack for the potential flow example problem and the example problem which includes the effect of the gas-dynamic boundary layer

5. METHODS OF ANALYSIS: THE DEICING FLUID

5.1 Assumptions

Before we can incorporate the deicing fluid into the programming that is complete on the potential flow and gas-dynamic boundary layer, we need to find the governing equation of motion for an element of fluid just as the Navier-Stokes equations and the Boundary Layer equations describe the motion of the gas-dynamic boundary layer.

Within the development of this equation of motion are several assumptions that we work with. The first of these is that we will initially ignore the non-linear effects of the deicing fluid. In other words, the deicing fluid will be assumed to have a constant dynamic viscosity. In future efforts, we believe the inclusion of this reality may be important. Specifically, we would have to solve the motion of the deicing fluid for an assumed value of the dynamic viscosity as a function of temperature and shear rate then compare the assumed value to that which has been experimentally determined for the calculated shear rate. Once the new dynamic viscosity is known, one can iterate on this procedure to account for the non-linearity of the fluid. When specific fluids are of interest, it will be important to model the shear stress-rate of strain relationship correctly.

Another assumption used in this study is that since the ratio of dynamic viscosities between the deicing fluid and the air is so large, inertia can be safely neglected in the analysis of the deicing fluid. The deicing fluids are generally very viscous with kinematic viscosities on the order of 200-400 cS and when air has a dynamic viscosity of about 1.98E-05 $N \cdot s/m^2$, the ratio of dynamic viscosities between the deicing fluid and the air is on the order of 12000-24000 assuming the density of the deicing fluid is similar to water. Again, the non-linearity or Non-Newtonian nature of the deicing fluid is difficult to quantify, but in general, the dynamic viscosities are much greater for the fluids than for the air. Because of the "high" dynamic viscosity, this is generally considered to be a "low Reynolds number flow" where the forces due to inertia are much less than those due to viscosity. Since this is true, inertia is neglected and terms involving local and convective accelerations are dropped from consideration in the development of the governing equation of motion for the deicing fluid.

The next assumption in this analysis is that the effects of gravity and surface tension can be ignored. As for gravity, it works in the vertical direction only and as we will show, we

do not require a conservation of momentum equation in the vertical direction. Surface tension would normally play a part in the analysis, but the realization of surface tension effects is to create a pressure differential across the free surface of the fluid between the outside of the deicing fluid and the inside of the deicing fluid. This pressure differential is inversely proportional to the radius of curvature of the deicing fluid surface. To consider these effects in the x momentum equation would require a change in this "jump" across the interface with the x coordinate. This would be a third-order effect and has negligible value in the analysis of forces in the x direction. It does, however become a concern if one were to look at the y component of momentum conservation as in a stability analysis.

The final assumption in this study is the lack of importance of fluid stability. There are ongoing investigations into the effects of fluid stability here at Iowa State and elsewhere, but the underlying assumption is the need for a baseline set of data or an unperturbed solution from which to begin the stability analysis. In a stability analysis, often the governing equations are linearized and a first order stability solution is obtained. A famous example of this is the Orr-Summerfield Equations; the linear stability analysis predicts regions of unstable flow for perturbations of various frequencies as a function of Reynolds number. Such analysis, while potentially valuable, is beyond the scope of this work as we will be interested only in developing the nominal solutions for the deicing fluid.

5.2 Development of the deicing fluid differential equation

With the assumptions listed in Section 5.1, we are ready to develop the differential equation of motion that describes the depth of the deicing fluid as a function of time and space. Taking a differential approach to the problem, a fluid element exists as shown in Figure 5.1. In this element, there are normal and tangential unit vectors, $\hat{n}$ and $\hat{t}$ respectively, at the free surface that have components parallel to the $\hat{i}$ and $\hat{j}$ unit vectors as follows:

$$\hat{n} = -\frac{\dfrac{\partial h}{\partial x}}{\sqrt{1+\left(\dfrac{\partial h}{\partial x}\right)^2}}\hat{i} + \frac{1}{\sqrt{1+\left(\dfrac{\partial h}{\partial x}\right)^2}}\hat{j} \tag{5.1}$$

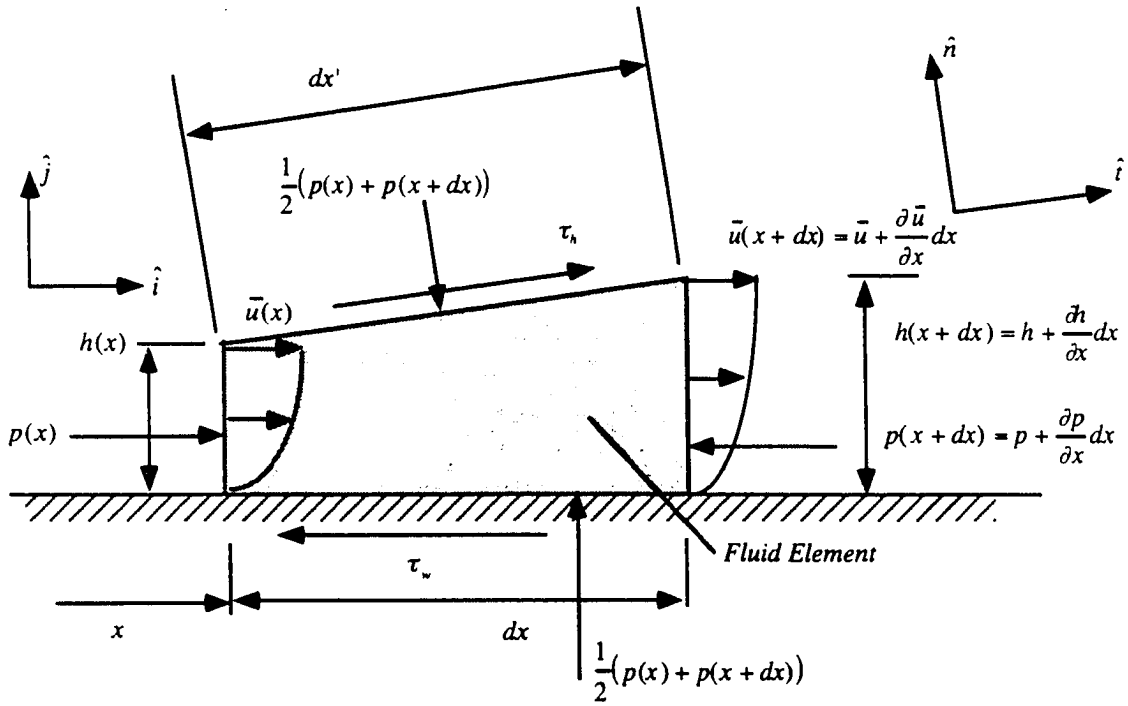


Figure 5.1: Fluid element definitions

$$\hat{t} = \frac{1}{\sqrt{1+\left(\frac{\partial h}{\partial x}\right)^2}}\hat{i} + \frac{\frac{\partial h}{\partial x}}{\sqrt{1+\left(\frac{\partial h}{\partial x}\right)^2}}\hat{j} \tag{5.2}$$

and as shown in Figure 5.1, the length,

$$dx' = dx\sqrt{1+\left(\frac{\partial h}{\partial x}\right)^2} \tag{5.3}$$

We assume in the two dimensional world that there is an incremental distance dz into the page through which there is no variation in any of the entities in this discussion. Acting on this fluid element are several forces that will shape the depth as a function of time. If the fluid element exists at a coordinate x and is of length dx in the x direction, neglecting inertia, we find a conservation of momentum equation in the x direction is appropriate. Namely,

$$\sum dF_x = 0 \tag{5.4}$$

$$= p[h\,dz] - \left(p + \frac{\partial p}{\partial x}dx\right)\left(h + \frac{\partial h}{\partial x}dx\right)dz$$

$$-\tau_w dx\,dz + \tau_h dx\sqrt{1+\left(\frac{\partial h}{\partial x}\right)^2}\,\frac{1}{\sqrt{1+\left(\frac{\partial h}{\partial x}\right)^2}}dz$$

$$+\frac{1}{2}(p(x)+p(x+dx))dx\sqrt{1+\left(\frac{\partial h}{\partial x}\right)^2}\,\frac{\frac{\partial h}{\partial x}}{\sqrt{1+\left(\frac{\partial h}{\partial x}\right)^2}}dz$$

where τ_h is the shear stress acting on the fluid at the interface, τ_w is the shear stress acting on the fluid from the wall and h is the deicing fluid depth at coordinate x.

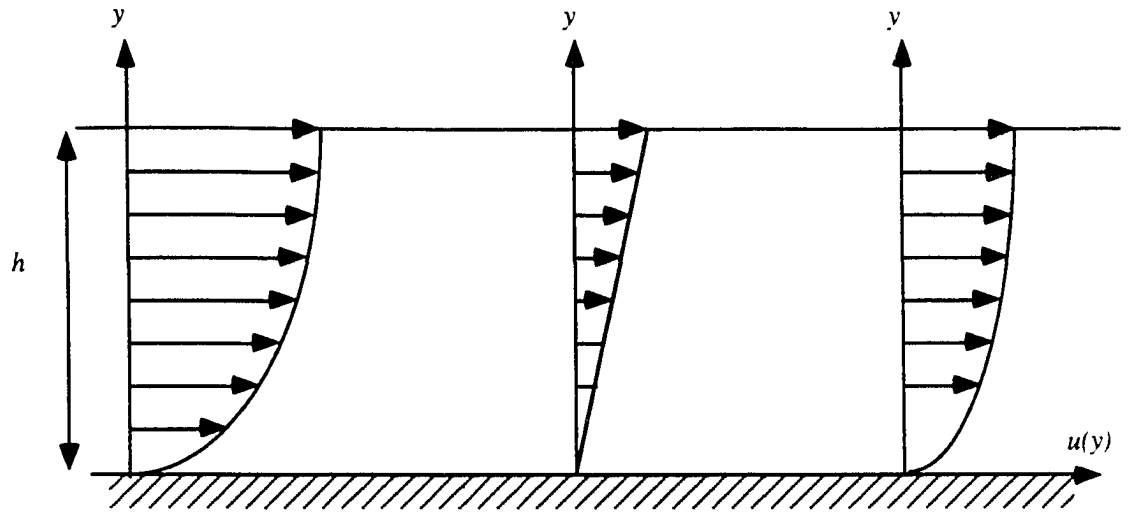
Equation (5.4) can be simplified. Note we have assumed that the pressure on the two "horizontal" faces of the fluid element are the average of the pressures $p(x)$ and $p(x+dx)$. Throughout this analysis, it is assumed that $\frac{\partial h}{\partial x} << 1$ and that $(dx)^2 << dx$. The simplification results in (including the neglect of "small" terms):

$$\frac{\partial p}{\partial x} = \frac{\tau_h - \tau_w}{h} \tag{5.5}$$

Now, suppose the velocity profile within the fluid were linear in the sense that individual velocity profiles can be superimposed to give the final velocity profile. This is a valid assumption considering we are neglecting inertia. Further, assume part of the velocity profile is due to a shear stress applied at the interface and part of the velocity profile is due to a pressure gradient. See Figure 5.2. If this is true, the generalized velocity profile at any station along the chord within the deicing fluid layer can be written as:

$$u(y) = a + by + cy^2 \tag{5.6}$$

Applying the boundary conditions that the shear stress at the wall is τ_w and the shear stress at the interface (free surface) is τ_h and that the velocity at $y = 0$ is zero (no-slip), we can solve for a, b, and c such that



Total velocity profile = *First part due to shear stress* + *Second part due to pressure pradient*

Figure 5.2: Linearity of velocity profile solution

$$u(y) = \frac{\tau_w}{\mu_f} y + \frac{\tau_h - \tau_w}{h2\mu_f} y^2 \tag{5.7}$$

assuming the deicing fluid is Newtonian in that the shear stress within the fluid is proportional to the velocity gradient with respect to y at that point and proportional to the dynamic viscosity of the deicing fluid, μ_f.

Performing an integration from $y = 0$ to $y = h$ will give us the average velocity, $\bar{u}$, as a function of the two shear stress values, the dynamic viscosity of the deicing fluid and the fluid depth.

$$\bar{u} = \frac{h}{6\mu_f}\left(2\tau_w + \tau_h\right) \tag{5.8}$$

Using Equation (5.5) to eliminate the shear stress at the wall gives us:

$$\bar{u} = \frac{\tau_h h}{2\mu_f} - \frac{\partial p}{\partial x}\frac{h^2}{3\mu_f} \tag{5.9}$$

One final component remains in this analysis, namely the conservation of mass for this incompressible fluid.

Taking the fluid element shown in Figure 5.1 and assuming the depth of the fluid can change with time, we might start with the integral representation of the conservation of mass equation.

$$\frac{\partial}{\partial t}\left(\int_{CV}\rho d\forall\right)+\int_{CS}\rho(\underline{V}\cdot\hat{n})dA=0 \tag{5.10}$$

the volume element $d\forall$, is written as

$$d\forall = dz\left(h\,dx+\frac{1}{2}dx\left(\frac{\partial h}{\partial x}dx\right)\right) \tag{5.11}$$

and taking the time-derivative of each of these terms (after eliminating the density and dz term, gives

$$\begin{aligned}\frac{\frac{\partial}{\partial t}\left(\int_{CV}\rho dz\left(h\,dx+\frac{1}{2}dx\left(\frac{\partial h}{\partial x}dx\right)\right)\right)}{\rho dz} &= \frac{\partial}{\partial t}(h\,dx)+\frac{\partial}{\partial t}\left(\frac{1}{2}dx\left(\frac{\partial h}{\partial x}dx\right)\right)\\ &= h\frac{\partial}{\partial t}(dx)+dx\frac{\partial h}{\partial t}\\ &\quad + \textit{negligible terms because } (dx)^2 << dx\\ &= \frac{\partial h}{\partial t}dx\end{aligned} \tag{5.12}$$

On to the mass flux term. Noticing there is no flow through either the top or bottom surface because of no-penetration and the definition of a free surface (the flow is tangential there), we get:

$$\frac{\int_{CS}\rho(\underline{V}\cdot\hat{n})dA}{\rho dz} = -\bar{u}h+(\bar{u}+\frac{\partial\bar{u}}{\partial x}dx)(h+\frac{\partial h}{\partial x}dx) \tag{5.13}$$

So the total mass conservation equation is written in differential form as:

$$\frac{\partial h}{\partial t}dx + -\bar{u}h + (\bar{u} + \frac{\partial \bar{u}}{\partial x}dx)(h + \frac{\partial h}{\partial x}dx) = 0 \tag{5.14}$$

$$\frac{\partial h}{\partial t}dx - \bar{u}h + \bar{u}h + \frac{\partial \bar{u}}{\partial x}hdx + \bar{u}\frac{\partial h}{\partial x}dx + terms\ of\ order\ dx^2 = 0$$

$$\frac{\partial h}{\partial t} + \bar{u}\frac{\partial h}{\partial x} + \frac{\partial \bar{u}}{\partial x}h = 0$$

$$\frac{\partial h}{\partial t} + \frac{\partial}{\partial x}(\bar{u}h) = 0$$

Now, replace $\bar{u}$ with its value from Equation (5.9) and we get, finally,

$$\frac{\partial h}{\partial t} + \frac{\partial}{\partial x}\left(\frac{\tau_h h^2}{2\mu_f} - \frac{\partial p}{\partial x}\frac{h^3}{3\mu_f}\right) = 0 \tag{5.15}$$

This is the differential equation of motion that requires numerical approximation for a complete solution to the deicing fluid depth as a function of time and one space coordinate.

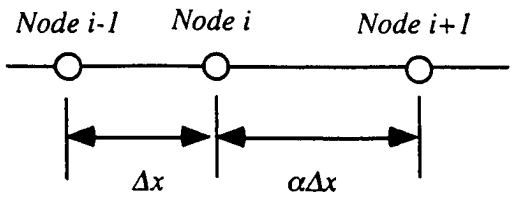
5.3 Numerical representation of the deicing fluid differential equation

The objective in this section is to determine a set of equations that might approximate the deicing fluid equation of motion. As in Chapter 4 of this work, and consistent with the assumptions listed as the beginning of this Chapter, the pressure gradient is assumed to be the same as in the boundary layer approximations to the gas-dynamic boundary layer. The error in making this approximation would be in neglecting the hydrostatic pressure due the density of the fluid (a gradient in the vertical direction) and any changes in the pressure due to surface tension (a third order effect). Therefore in the approximation of Equation (5.15), the pressure gradient is approximated to be:

$$\frac{\partial p}{\partial x} = -\rho\left(\frac{\partial V_e}{\partial t} + V_e\frac{\partial V_e}{\partial x}\right) \tag{5.16}$$

With this in mind, the deicing fluid equation of motion becomes:

$$\frac{\partial h}{\partial t} + \frac{\partial}{\partial x}\left(\frac{\tau_h h^2}{2\mu_f} + \rho\left(\frac{\partial V_e}{\partial t} + V_e\frac{\partial V_e}{\partial x}\right)\frac{h^3}{3\mu_f}\right) = 0 \tag{5.17}$$



Note: Node i corresponds to the midpoint of panel i

Figure 5.3: Deicing fluid discretization convention

One of the first things one notices about Equation (5.17) is that it is non-linear in h. This will be discussed in the next section, but for the moment, let us focus on a numerical approximation to this equation via finite differences.

First of all, let a dummy variable, z, be defined such that:

$$z = \frac{\tau_h h^2}{2\mu_f} + \rho\left(\frac{\partial V_e}{\partial t} + V_e \frac{\partial V_e}{\partial x}\right)\frac{h^3}{3\mu_f} \tag{5.18}$$

So now our differential equation of motion becomes:

$$\frac{\partial h}{\partial t} + \frac{\partial}{\partial x}(z) = 0 \tag{5.19}$$

Let a first order approximation to the time derivative be

$$\frac{\partial h}{\partial t} = \frac{h_i^{k+1} - h_i^k}{\Delta t} \tag{5.20}$$

where the subscript i on the depth, h, refers to the depth at node i defined by Figure 5.3. The time step is designated by the superscript (k and k+1) where we are searching for a solution at time step k+1.

Assume that the deicing fluid is initially applied on the top surface of the airfoil at depth of h_o and begins at a position x_f (See Chapter 4) and continues to the trailing edge. The coordinate x_f will be close to a node that is defined as *ifluid* within the programming. With the uneven spacing shown in Figure 5.3 (the deicing fluid module nodes correspond to the midpoint of each panel where the edge velocities are known from the potential flow solution and the shear stresses at the interface are known from the gas-dynamic boundary layer solution), we approximate the second term in Equation (5.19) by:

$$\frac{\partial}{\partial x}(z) = \frac{z_{i+1}^{k+1} + (\alpha^2 - 1)z_i^{k+1} - \alpha^2 z_{i-1}^{k+1}}{\alpha(\alpha+1)\Delta x} \tag{5.21}$$

where

$$z_{i+1}^{k+1} = \frac{\tau_{h,i+1}^{k+1}\left(h_{i+1}^{k+1}\right)^2}{2\mu_f} + \left[\rho\left(\frac{V_{e,i+1}^{k+1} - V_{e,i+1}^{k}}{\Delta t} + V_{e,i+1}^{k+1}\left(\frac{V_{e,i+1}^{k+1} - V_{e,i}^{k+1}}{\alpha\Delta x}\right)\right)\frac{\left(h_{i+1}^{k+1}\right)^3}{3\mu_f}\right] \tag{5.22}$$

$$z_i^{k+1} = \frac{\tau_{h,i}^{k+1}\left(h_i^{k+1}\right)^2}{2\mu_f} + \left[\rho\left(\frac{V_{e,i}^{k+1} - V_{e,i}^{k}}{\Delta t} + V_{e,i}^{k+1}\left(\frac{V_{e,i+1}^{k+1} + (\alpha^2 - 1)V_{e,i}^{k+1} - \alpha^2 V_{e,i-1}^{k+1}}{\alpha(\alpha+1)\Delta x}\right)\right)\frac{\left(h_i^{k+1}\right)^3}{3\mu_f}\right] \tag{5.23}$$

and

$$z_{i-1}^{k+1} = \frac{\tau_{h,i-1}^{k+1}\left(h_{i-1}^{k+1}\right)^2}{2\mu_f} + \left[\rho\left(\frac{V_{e,i-1}^{k+1} - V_{e,i-1}^{k}}{\Delta t} + V_{e,i-1}^{k+1}\left(\frac{V_{e,i}^{k+1} - V_{e,i-1}^{k+1}}{\Delta x}\right)\right)\frac{\left(h_{i-1}^{k+1}\right)^3}{3\mu_f}\right] \tag{5.24}$$

Simply put, given the initial depth of h_o spread over the airfoil from x_f to the trailing edge, and the input edge velocities and shear stresses, we approximate the change in depth of the deicing fluid with time through the non-linear approximations to Equation (5.19). The method is implicit in nature, again in an attempt to avoid numerical instability. The fluid depths at h_i^{k+1} are solved for all at once (for all stations along the airfoil) and the procedure moves to the next time step. As will be discussed in the section on "Special Considerations", the two endpoints, *h* at *ifluid* and station *NN+1* will require special attention. The problem becomes how do we solve a series of equations for i = *ifluid+1* to *NN* to find h_i^{k+1} for the following final approximation to the deicing fluid equation of motion?

$$\frac{h_i^{k+1}-h_i^k}{\Delta t}+\frac{1}{\alpha(\alpha+1)\Delta x}\left[\begin{array}{c}\dfrac{\tau_{h,i+1}^{k+1}\left(h_{i+1}^{k+1}\right)^2}{2\mu_f}+\\ \left[\rho\left(\dfrac{V_{e,i+1}^{k+1}-V_{e,i+1}^{k}}{\Delta t}+V_{e,i+1}^{k+1}\left(\dfrac{V_{e,i+1}^{k+1}-V_{e,i}^{k+1}}{\alpha\Delta x}\right)\right)\dfrac{\left(h_{i+1}^{k+1}\right)^3}{3\mu_f}\right]\end{array}\right]$$

$$+\frac{(\alpha^2-1)}{\alpha(\alpha+1)\Delta x}\left[\begin{array}{c}\dfrac{\tau_{h,i}^{k+1}\left(h_{i}^{k+1}\right)^2}{2\mu_f}+\\ \left[\rho\left(\dfrac{V_{e,i}^{k+1}-V_{e,i}^{k}}{\Delta t}+V_{e,i}^{k+1}\left(\dfrac{V_{e,i+1}^{k+1}+(\alpha^2-1)V_{e,i}^{k+1}-\alpha^2V_{e,i-1}^{k+1}}{\alpha(\alpha+1)\Delta x}\right)\right)\dfrac{\left(h_{i}^{k+1}\right)^3}{3\mu_f}\right]\end{array}\right]$$

$$-\frac{\alpha^2}{\alpha(\alpha+1)\Delta x}\left[\begin{array}{c}\dfrac{\tau_{h,i-1}^{k+1}\left(h_{i-1}^{k+1}\right)^2}{2\mu_f}+\\ \left[\rho\left(\dfrac{V_{e,i-1}^{k+1}-V_{e,i-1}^{k}}{\Delta t}+V_{e,i-1}^{k+1}\left(\dfrac{V_{e,i}^{k+1}-V_{e,i-1}^{k+1}}{\Delta x}\right)\right)\dfrac{\left(h_{i-1}^{k+1}\right)^3}{3\mu_f}\right]\end{array}\right]=0 \qquad (5.25)$$

To be consistent in the study of the numerical representation of the deicing fluid equation of motion, one needs to account for the changing of the x coordinate with time as was done in the solution of the gas-dynamic boundary layer . Here, computational coordinates are ζ and τ and we should represent the time derivative of the h by:

$$\frac{\partial h}{\partial t}=\frac{\partial h}{\partial \tau}+\frac{\partial \zeta}{\partial t}\frac{\partial h}{\partial \zeta}=\frac{h_i^{k+1}-h_i^k}{\Delta\tau}+\frac{\zeta_i^{k+1}-\zeta_i^k}{\Delta\tau}\left(\frac{h_{i+1}^{k+1}+(\alpha^2-1)h_i^{k+1}-\alpha^2h_{i-1}^{k+1}}{\alpha(\alpha+1)\Delta\zeta}\right) \qquad (5.26)$$

so that in computational coordinates, the final set of equations for $i = ifluid+1, \ldots, NN$ are:

$$\frac{h_i^{k+1}-h_i^k}{\Delta\tau}+\frac{\zeta_i^{k+1}-\zeta_i^k}{\Delta\tau}\left(\frac{h_{i+1}^{k+1}+(\alpha^2-1)h_i^{k+1}-\alpha^2 h_{i-1}^{k+1}}{\alpha(\alpha+1)\Delta\zeta}\right) \tag{5.27}$$

$$+\frac{1}{\alpha(\alpha+1)\Delta\zeta}\left[\frac{\tau_{h,i+1}^{k+1}\left(h_{i+1}^{k+1}\right)^2}{2\mu_f}+\left[\rho\left(\frac{V_{e,i+1}^{k+1}-V_{e,i+1}^{k}}{\Delta\tau}+V_{e,i+1}^{k+1}\left(\frac{V_{e,i+1}^{k+1}-V_{e,i}^{k+1}}{\alpha\Delta\zeta}\right)\right)\frac{\left(h_{i+1}^{k+1}\right)^3}{3\mu_f}\right]\right]$$

$$+\frac{(\alpha^2-1)}{\alpha(\alpha+1)\Delta\zeta}\left[\frac{\tau_{h,i}^{k+1}\left(h_i^{k+1}\right)^2}{2\mu_f}+\left[\rho\left(\frac{V_{e,i}^{k+1}-V_{e,i}^{k}}{\Delta\tau}+V_{e,i}^{k+1}\left(\frac{V_{e,i+1}^{k+1}+(\alpha^2-1)V_{e,i}^{k+1}-\alpha^2 V_{e,i-1}^{k+1}}{\alpha(\alpha+1)\Delta\zeta}\right)\right)\frac{\left(h_i^{k+1}\right)^3}{3\mu_f}\right]\right]$$

$$-\frac{\alpha^2}{\alpha(\alpha+1)\Delta\zeta}\left[\frac{\tau_{h,i-1}^{k+1}\left(h_{i-1}^{k+1}\right)^2}{2\mu_f}+\left[\rho\left(\frac{V_{e,i-1}^{k+1}-V_{e,i-1}^{k}}{\Delta\tau}+V_{e,i-1}^{k+1}\left(\frac{V_{e,i}^{k+1}-V_{e,i-1}^{k+1}}{\Delta\zeta}\right)\right)\frac{\left(h_{i-1}^{k+1}\right)^3}{3\mu_f}\right]\right]=0$$

5.4 Solution of a non-linear equation

The solution of a set of equations such as those shown in Equation (5.27) requires special attention. The usual method is to resort to iterative methods to the solve for the variables, h_i^{k+1}. One such method, and the one employed in this study, is Newton's method. The procedure for this is as follows.

First, determine if the current predicted values for h_i^{k+1} satisfy the set of equations (Equation (5.27)). Most of the time, they do not and there is some "residual" value that is left over. For the current approximation to h_i^{k+1}, we calculate the left hand side of Equation (5.27). We then replace zero on the right hand side of this equation and set that equal to the residual for the current approximation. We will call these f_i. The objective is to choose the values for h_i^{k+1} such that the maximum value of the individual residuals is minimized. So, the first step in Newton's method is to calculate the residuals. If the maximum residual is small enough, we consider the solution to be appropriate and move on to the next time step. If the maximum

residual is not small enough, then we need to figure out the best path to minimize the individual values of the residuals. The next step, therefore, is to calculate a square Jacobain matrix, $\underline{\underline{J}}$, (with an index that goes from *ifluid+1* to *NN*) such that:

$$J_{ij} = \frac{\partial f_i}{\partial h_j^{k+1}} \tag{5.28}$$

The objective is to determine a set of correction values, $\underline{dh}$, such that when we add these to the last approximation for h_i^{k+1} we get closer to minimizing the residual values. In our problem, we get for the components of the Jacobian:

$$J_{i,i-1} = \frac{\zeta_i^{k+1} - \zeta_i^k}{\Delta\tau}\left(\frac{-\alpha^2}{\alpha(\alpha+1)\Delta\zeta}\right) - \frac{\alpha^2}{\alpha(\alpha+1)\Delta\zeta}\left[\begin{array}{c} \dfrac{\tau_{h,i-1}^{k+1}\left(h_{i-1}^{k+1}\right)}{\mu_f} + \\ \left[\rho\left(\dfrac{V_{e,i-1}^{k+1} - V_{e,i-1}^k}{\Delta\tau} + V_{e,i-1}^{k+1}\left(\dfrac{V_{e,i}^{k+1} - V_{e,i-1}^{k+1}}{\Delta\zeta}\right)\right)\dfrac{\left(h_{i-1}^{k+1}\right)^2}{\mu_f}\right] \end{array}\right] \tag{5.29}$$

$$J_{i,i} = \frac{1}{\Delta\tau} + \frac{\zeta_i^{k+1} - \zeta_i^k}{\Delta\tau}\left(\frac{(\alpha^2-1)}{\alpha(\alpha+1)\Delta\zeta}\right) + \frac{(\alpha^2-1)}{\alpha(\alpha+1)\Delta\zeta}\left[\begin{array}{c} \dfrac{\tau_{h,i}^{k+1}\left(h_i^{k+1}\right)}{\mu_f} + \\ \left[\rho\left(\dfrac{V_{e,i}^{k+1} - V_{e,i}^k}{\Delta\tau} + V_{e,i}^{k+1}\left(\dfrac{V_{e,i+1}^{k+1} + (\alpha^2-1)V_{e,i}^{k+1} - \alpha^2 V_{e,i-1}^{k+1}}{\alpha(\alpha+1)\Delta\zeta}\right)\right)\dfrac{\left(h_i^{k+1}\right)^2}{\mu_f}\right] \end{array}\right] \tag{5.30}$$

and

$$J_{i,i+1} = \frac{1}{\Delta\tau} + \frac{\zeta_i^{k+1} - \zeta_i^k}{\Delta\tau}\left(\frac{1}{\alpha(\alpha+1)\Delta\zeta}\right) + \frac{1}{\alpha(\alpha+1)\Delta\zeta}\left[\begin{array}{c} \dfrac{\tau_{h,i+1}^{k+1}\left(h_{i+1}^{k+1}\right)}{\mu_f} + \\ \left[\rho\left(\dfrac{V_{e,i+1}^{k+1} - V_{e,i+1}^k}{\Delta\tau} + V_{e,i+1}^{k+1}\left(\dfrac{V_{e,i+1}^{k+1} - V_{e,i}^{k+1}}{\alpha\Delta\zeta}\right)\right)\dfrac{\left(h_{i+1}^{k+1}\right)^2}{\mu_f}\right] \end{array}\right] \tag{5.31}$$

Now that the terms are known in the Jacobian, we go to the next step of Newton's Method, namely finding the correction values, dh_i for $i = ifluid+1$ to NN. The Jacobian, being a tridiagonal matrix, is used in the following sets of equations:

$$\underline{\underline{J}}\,\underline{dh} = -\underline{f} \tag{5.32}$$

We call on the Thomas Algorithm again (taking advantage of the tridiagonal nature of the Jacobian) to solve for the correction values. Continuing this procedure, we zero in on the values for h_i^{k+1} usually within one or two iterations. The time-consuming part of this process is not the iteration itself, but rather the calculation of the residuals and the Jacobian matrix. We now have a differential equation of motion for the deicing fluid, a numerical approximation to this equation of motion and a method of solution. There are several items which deserve special attention, though, as will be seen in the next section.

5.5 Special considerations

As mentioned previously, the two endpoints in the above analysis, namely at $i = ifluid$ and at $i = NN+1$ (the nodes go to $NN+1$ to coincide with the numbering of the gas-dynamic boundary layer analysis which includes one more node than panels because of the inclusion of the stagnation point). In this analysis, those two points are handled using one-sided approximations to the space derivatives. The solution is still found using an iterative method on the values at the endpoints, but as an example, for $i = ifluid$, with $\Delta\zeta = \zeta_{ifluid+1}^{k+1} - \zeta_{ifluid}^{k+1}$ we find the following numerical formulation of the residual.

$$\frac{h_{ifluid}^{k+1}-h_{ifluid}^{k}}{\Delta\tau}+\frac{\zeta_{ifluid}^{k+1}-\zeta_{ifluid}^{k}}{\Delta\tau}\left(\frac{h_{ifluid+1}^{k+1}-h_{ifluid}^{k+1}}{\Delta\zeta}\right) \tag{5.33}$$

$$+\frac{1}{\Delta\zeta}\left[\left[\rho\left(\frac{V_{e,ifluid+1}^{k+1}-V_{e,ifluid+1}^{k}}{\Delta\tau}+V_{e,ifluid+1}^{k+1}\left(\frac{V_{e,ifluid+1}^{k+1}-V_{e,ifluid}^{k+1}}{\Delta\zeta}\right)\right)\frac{\left(h_{ifluid+1}^{k+1}\right)^{3}}{3\mu_f}+\frac{\tau_{h,ifluid+1}^{k+1}\left(h_{ifluid+1}^{k+1}\right)^{2}}{2\mu_f}\right]\right]$$

$$-\frac{1}{\Delta\zeta}\left[\left[\rho\left(\frac{V_{e,ifluid}^{k+1}-V_{e,ifluid}^{k}}{\Delta\tau}+V_{e,ifluid}^{k+1}\left(\frac{V_{e,ifluid+1}^{k+1}-V_{e,ifluid}^{k+1}}{\Delta\zeta}\right)\right)\frac{\left(h_{ifluid}^{k+1}\right)^{3}}{3\mu_f}+\frac{\tau_{h,ifluid}^{k+1}\left(h_{ifluid}^{k+1}\right)^{2}}{2\mu_f}\right]\right]=f_{ifluid}$$

The Jacobian is found by taking the partial derivative with respect to h_{ifluid}^{k+1} and the same iterative procedure as outlined above is used. In likewise manner, we find the values for h_{NN+1}^{k+1}. So, prior to moving to the next time step, all of the fluid depths are known.

A second consideration in the solution of the deicing fluid depth as a function of time and space comes when the values for the depth come very close to zero. Because of numerical inaccuracies due to discretization and truncation error in the formulation of the finite difference representations of the deicing fluid equation of motion, on occasion, at very small fluid depths (on the order of 10^{-6} (non-dimensionalized)) the program may compute a depth which is a very small negative number. We know that this is not possible, physically, so there is a flag in the programming that as the depth becomes negative, the all corrections should make the depth increase. Usually, this increase is of the same order of magnitude (up to positive 10^{-6}). Once the fluid has a small positive depth again, the flag is removed and the solution continues its normal progression. This condition then allows for an oscillation about zero for the point in question. As such, a solution that iterates beyond one or two iterations is usually indicative of this oscillating nature. Therefore, there is another flag in the solution that places a maximum limit on the number of iterations (currently set at 20). Much work has verified this oscillating feature of the solution. This special consideration is used in the programming without further mention.

5.6 General procedure for solving the full problem

Now, a general outline is in order to understand how the complete program might work. The integration of all three modules of the programming, the potential flow solution, the solution of the gas-dynamic boundary layer and the deicing fluid equation of motion module, will allow us to look at the effect of the deicing fluid on the aerodynamic performance of a two-dimensional airfoil on general aviation aircraft. We will wait until the next Chapter to make general observations and analysis.

1. The first step in all of the problems is to discretize the airfoil using the cosine distribution mentioned in Chapter 3. This gives the number of panels, *NN*. The endpoints and midpoints of each panel are defined in this step. The spacing of the gas-dynamic boundary layer grid is input prior to analysis. Included here are the number of nodes in the vertical direction at each station, the region of interest (as defined in Chapter 4), and the stretching factor, β.
2. If there is deicing fluid present, the place for the deicing fluid to begin on the airfoil is input as well as the initial uniform depth. The ratio of dynamic viscosities between the deicing fluid and the air is also input at this point.
3. The control parameters for the take-off simulation are input prior to the analysis. The initial angle of attack, the pitch rate, the initial velocity, and freestream acceleration rates during the take-off simulation are given.
4. Prior to the analysis of the gas-dynamic boundary layer, the airfoil is allowed to develop a steady-state pressure distribution at the initial speed. The airfoil starts impulsively from zero to the initial speed developing a bound vortex. This bound vortex is shed and a new pressure distribution is determined. This process of shedding vortices continues until the pressure distribution does not change much from one time step to the next.
5. The next step is to determine a steady-state development of the gas-dynamic boundary layer that includes the effect of a displacement thickness. As shown in Appendix A, this is an iterative process. Take two time steps, k and $k+1$. The values for the components of velocity should be the same for both of these time steps. We might know what the stagnation point flow looks like, but as we move, say, to the top of the airfoil, we predict a velocity distribution at the station after the stagnation point for time step $k+1$. More than likely, this is not the same as the velocity distribution at that station from the previous time step, so we set the velocity distribution at the previous

time step at that station equal to the current prediction for the velocity distribution. Eventually, the two velocity distributions agree (within an arbitrary tolerance) and we can move to the next station. This process is continued until the entire gas-dynamic boundary layer is mapped out for the steady-state case. Once this steady-state is determined, the normal velocities on the panels are found to model the effect of displacement thickness. Naturally, this modifies the outer flow. Once the normal velocities are found, a new potential flow is calculated and then the boundary layer flow is calculated again until the normal velocities due to the displacement thickness effects do not differ by a specified tolerance.

Throughout this process, the inclusion of the deicing fluid will modify the potential flow in the following manner. The idea behind the inclusion of displacement thickness effects is that it "pushes" the flow away from the airfoil. In a like way, we believe the deicing fluid to have the same effect as the displacement thickness. The condition for which we account for displacement thickness is now modified to include the deicing fluid via:

$$V_n = \frac{\partial}{\partial x}\left(V_e\left(\delta^* + h\right)\right) \tag{5.34}$$

In the steady-state development of the boundary layer, the motion of the deicing fluid is neglected. The initial depth is thought to exist at both of the steady time steps. Even if it was included, the motion would be minimal as the shear stresses and pressure gradients are small at this speed.

6. Once the steady-flow solution is found, we wish to move to the next time step. Initially, the distribution of normal velocities on the panel are found to be in proportion to the ratio of the current freestream velocity (at time step $k+1$) to the last freestream velocity and the distribution of normal velocities from the last time step. Then, assuming the no-slip condition in the boundary layer provides for zero horizontal velocity at the interface of the deicing fluid and gas-dynamic boundary layer, the gas-dynamic boundary layer is found for the given pressure distribution. This gives a distribution of shear stress which, when coupled with the current pressure distribution creates the motion of the deicing fluid. We know that the motion of the deicing fluid provides for a horizontal slip velocity at the interface of the deicing fluid and gas-dynamic boundary layer. Once this slip velocity is found, it is put back into the gas-dynamic boundary module. The gas-dynamic boundary layer is recalculated and this process of iteration between the gas-dynamic boundary layer and deicing fluid is continued until the shear stress at the interface and the slip velocity are essentially

equivalent between the two, thus the kinematic and dynamic boundary conditions are matched at the interface.

7. Once the matching conditions at the interface are met, we calculate a new distribution of normal velocities on the panels via Equation (5.34). We iterate on this process until the normal velocities are consistent from one iteration to the next. This process usually takes two to three chances at finding an appropriate distribution of normal velocities before satisfaction is reached. With this condition met, we are ready to proceed to the next time step.

5.7 Example problem results

Prior to a full discussion of the results in the next Chapter, it would be useful to carry our example problem to the end. For deicing fluid that is placed on top of the airfoil from 1% of the leading edge to the trailing edge at an initial depth of 0.001 (non-dimensionalized to the chord length) with a dynamic viscosity that is 24000 times that of the air, we get the following results as shown in Figure 5.4. Shown in Figure 5.4 is the results from the clean wing under the same take-off simulation as well. It appears that there is not much difference in the maximum lift coefficient prior to separation. Figure 5.5 shows a close-up view of the results near separation. This may or may not be important, but emphasizes the need for close attention in the region near separation.

The separation point for the clean wing was 15.719 degrees ($c_l = 1.5700$) while for the contaminated wing, separation was found to be at 15.299 degrees ($c_l = 1.487$). This difference is very small and might be attributed to how the time-steps near separation are handled. On the other hand, separation in the clean wing case was shown to be at 99.4% of chord while the contaminated wing had a separation at 98.7% of chord. It is expected that poorer performance would be indicated by separation further toward the leading edge. In any case, for this test, the results were not enough to develop a robust theory as to the cause of performance loss.

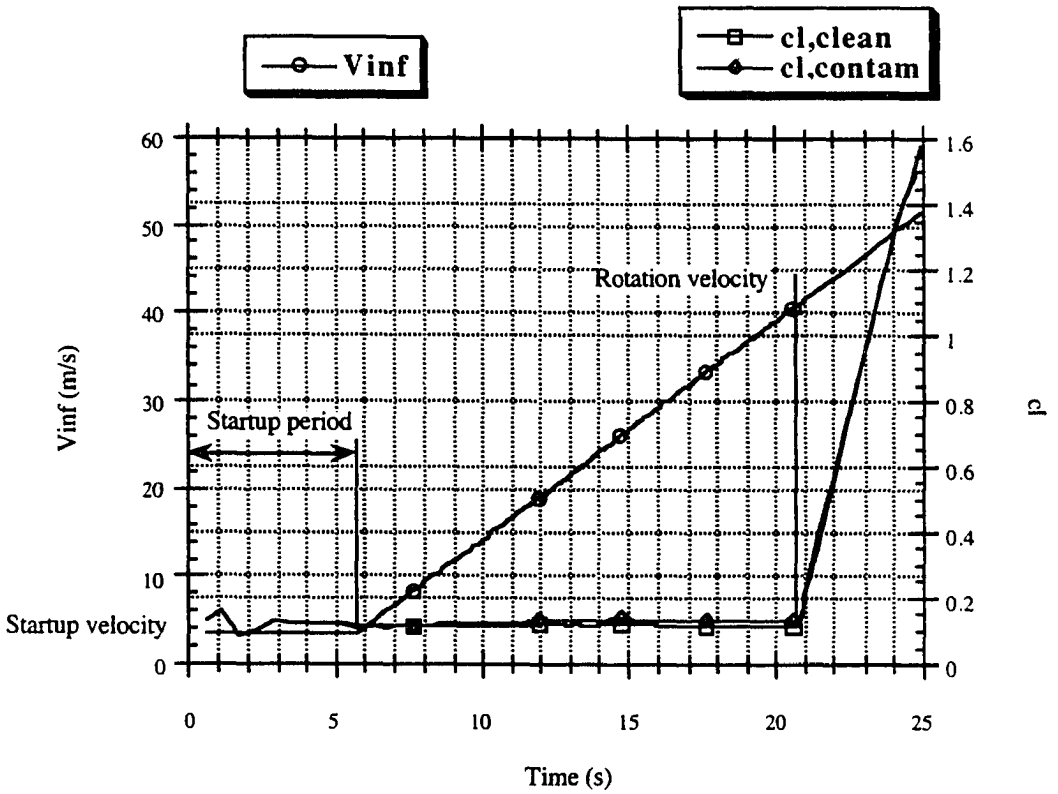


Figure 5.4: Performance results of the example problem with deicing fluid applied

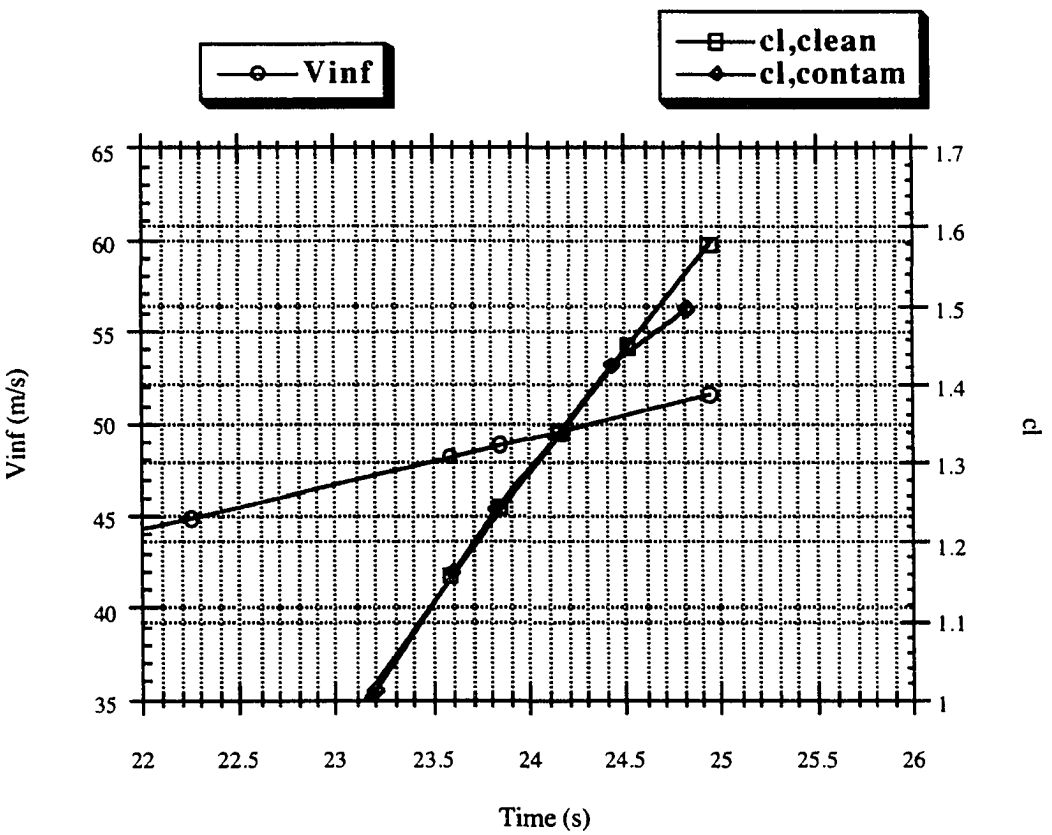


Figure 5.5: Close-up of lift coefficients near separation for the example problem

6. RESULTS

6.1 Discussion of time-steps required

Prior to looking at some general results, it will be useful to look at the choices that were made concerning the time steps used to solve the various problems we will present.

Normally, there are not a lot of numerical restrictions on the time steps used using implicit methods of solution. All of the various numerical schemes implemented in this study are of "first order". We would expect that for smaller time steps, the "accuracy" of the estimates would be better. Following this reasoning, we would like to use smaller and smaller time steps. Unfortunately, there is a limit to this thought process in this problem. This is realized in how the shed vortices react with the potential flow solutions near the trailing edge. With the Kutta Condition implemented in this problem as a matching of tangential velocities at the first and last panels (the ones closest to the trailing edge on bottom and top of the airfoil respectively), if the most recently shed vortex (modeled as a point vortex) is "too close" to the trailing edge, not only is the outer flow in the potential flow solution modified adversely, but in turn, this effects the boundary layer development and ultimately, the deicing fluid motion (see Figure 3.6 repeated here for convenience).

Take a shed vortex that is of magnitude Γ in the counter-clockwise direction that leaves the trailing edge. According to the model implemented in the time-dependent potential flow solution, this point vortex moves a distance $V_\infty \Delta t$ parallel to the line that bisects the upper and lower surface at the trailing edge. If Δt is small, the normal velocities on the panels near the trailing edge can be quite large. To avoid this, within the programming, the time step was maintained as the maximum of two parameters. During the take-off roll when the magnitude of the shed vortices is small, the time step is set such that any trailing point vortex lines up two chord lengths behind the trailing edge. This means that the time step is inversely proportional to the freestream velocity. During rotation, when the shed vortices can become large, the time step is proportional to the magnitude of the shed vortex so as to allow the most recently shed vortex enough time to move away so as to not influence the solution adversely. The constant of proportionality was chosen so that the transition point within the gas-dynamic boundary layer did not change by more than one "x" station along the chord on the bottom of the airfoil from one time step to the next. If this were true, it was assumed that the trailing vortex was

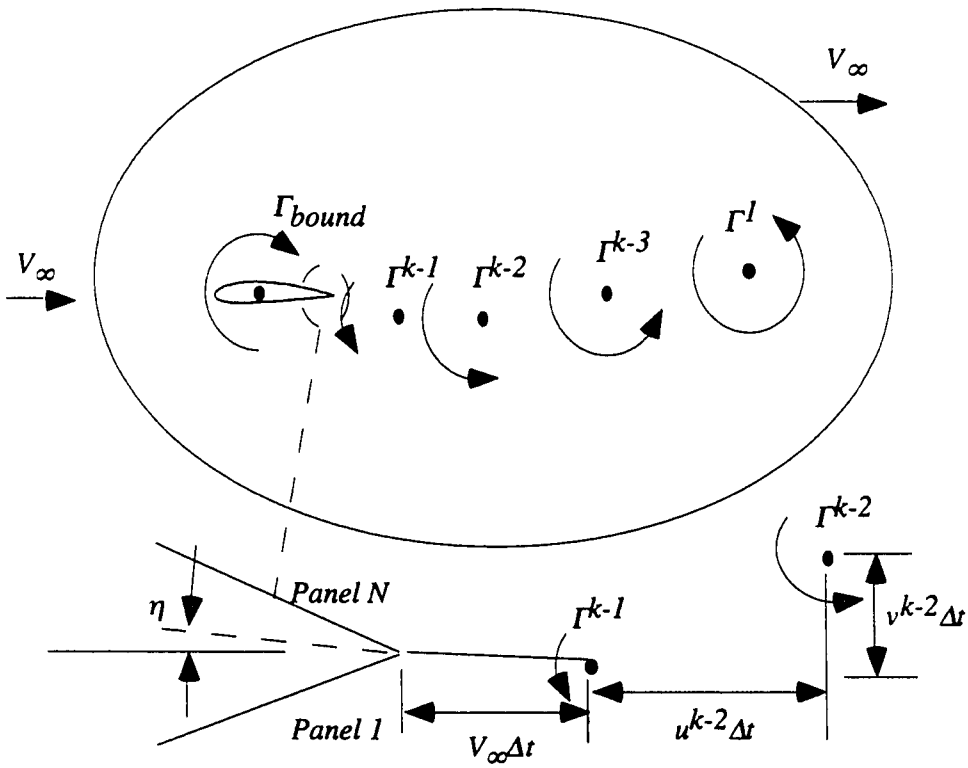


Figure 3.6: Drifting away of the starting vortices and transport of the vortices

unduly interfering with the solution. During the rotation maneuver, when the lift is increasing and hence the bound circulation and shed vortex strength must increase, this leads to rather large time steps. As stated before, large time steps tend to decrease the accuracy of the approximation to the governing equations. As the rotation continues to higher and higher angles of attack, the leading edge stagnation point moves below the leading edge and the transition point for the bottom gas-dynamic boundary layer moves toward the trailing edge. By the time the airfoil reaches its maximum lift, the lower transition point is almost to the trailing edge for the problems examined within this study.

With the time steps increasing due to the increased shed vortex size, it turns out that this can be a problem for the deicing fluid module. During the rotation, the pressure gradients and shear stresses within the gas-dynamic boundary layer are at their largest compared the take-off roll because of increased freestream values and an increasing angle of attack. Recall Equation 5.15,

$$\frac{\partial h}{\partial t} + \frac{\partial}{\partial x}\left(\frac{\tau_h h^2}{2\mu_f} - \frac{\partial p}{\partial x}\frac{h^3}{3\mu_f}\right) = 0 \tag{5.15}$$

which describes the motion of the deicing fluid. If the second term is "large", then in order for the first term to counteract this, either the time step should be small or the change in fluid depth from one time step to the next should be large. There are regions which develop large deviations in h as a function of x (aft of the transition point, in particular, where the shear stress increases quickly in space pulling the fluid with it) and can produce large values within the second term. In looking at Equation 5.15, it appears that this is like a one-dimensional wave equation (parabolic in nature if the term within the parentheses is replaced by a single variable dependent on h). With a fully implicit approach such as we have taken in the numerical estimate of the deicing fluid equation, there is no chance of numerical instabilities. However, using an approach such as

$$\frac{\partial h}{\partial t} + K\frac{\partial}{\partial x}(h) = 0 \approx \frac{h_i^{k+1} - h_i^k}{\Delta t} + K\left(\frac{h_{i+1}^{k+1} - h_{i-1}^{k+1}}{2\Delta x}\right) \tag{6.1}$$

where k is for the time step and i is used to designate spatial nodes, we are left with a truncation error as follows [1]:

$$\frac{\partial h}{\partial t} + K\frac{\partial}{\partial x}(h) = \left(\left[\frac{1}{2}K^2\Delta t\right]\frac{\partial^2 h}{\partial x^2} - \left[\frac{1}{6}K(\Delta x)^2 + \frac{1}{3}K^3(\Delta t)^2\right]\frac{\partial^3 h}{\partial x^3} + \cdots\right) \tag{6.2}$$

This means that even though we wish to avoid numerical instability with an implicit approach, if the second derivative of h with respect to position is large (as in the case of high - frequency waves), we need to maintain small time steps to insure accuracy. So with large time steps and high second derivatives of h, we would expect poor solutions. As such, we need to make the time steps small enough as to increase accuracy.

Therefore, within the deicing fluid module, for a total time interval dependent on the solution of the potential flow and gas-dynamic boundary layer, we break this interval into smaller time steps for the deicing fluid taking mini-time steps of 0.02 seconds with the last mini-time step being the remainder greater than 0.02 but less than 0.04 seconds. This method was more successful in capturing the "wave-like" patterns of the deicing fluid after rotation.

One other modification to the time step was made in this study. During the rotation maneuver, the shed circulation can build up rather quickly if the pitch rate is high enough. The difference between the bound vortex and the sum of the previously shed trailing vortices in the field can become great requiring even larger shed vortices and longer time steps based on the

criteria in the preceding paragraphs. With this, there is the potential to "overshoot" the actual point of first separation if the time steps are too large. To get around this, a method used consistently throughout the study was tested and implemented with success.

This author is not aware of any theories as to why this method is applicable other than the fact that it did not change the outer flow more than what would be expected due to the time dependence. As discussed in the RECOMMENDATIONS section of this dissertation, this tool is considered to be the weakest link in the study.

As the flow nears separation, the time steps are increasing based on the shed vortex criteria described above. To insure that we don't overshoot the separation point, once the time step based on the shed vortex criteria predicts an angle of attack greater than 11 degrees, the time step is modified to go to the point where the angle of attack is closest to 11 degrees (the assumption here is that the flow is near separation, but not quite separated) at time step k. This time step is completed and the vortex that should be shed after the 11 degree AOA solution is placed at the position of the shed point vortex from the previous time step (k-1). Thus the point vortex from time step k-1 has a larger magnitude, but is further from the trailing edge. The next time step (k+1) is set by the criteria based on the total circulation of the new point vortex from time step (k-1) and the shed vortex strength from time step k is set to zero. Thus circulation is still conserved, but the time steps are reasonable and the potential for overshoot is minimized.

The real problem in working with all these changes in time-step is the fact that there are two different types of problems that are being matched at the interface of the deicing fluid. Outside the gas-dynamic boundary layer is a high Reynolds number flow where characteristic velocities are on the order of the kinematic viscosity divided by a reference length (a small number) while in the deicing fluid, the characteristic velocities are of a much higher value. This means that to travel a constant distance, the air requires much less time than the deicing fluid. At the interface, these two velocities are matched and conflicting in their physical assumptions. This appears to be part of the difficulty in using the appropriate time-step size.

6.2 Fluid depth distributions

6.2.1 Fluid depth time-history

Prior to looking at the global results, it might be interesting to look at some distributions of deicing fluid depth as a function of time. One of the most insightful aspects of this research was the time history of the deicing fluid depth along the chord. A typical time-

history is shown in Figure 6.1. As time increases, two distinct regions are important. This is the deicing fluid depth for the example problem in Chapter 5 (a rotation speed of 41.0 m/s and a 3.5 degrees per second pitch rate), except that the ratio of dynamic viscosities between the deicing fluid and the air is 12000 rather than 24000 as was used in the example problem. Prior to rotation at about 20.97 seconds, not much happens. The fluid starts at a uniform non-dimensional depth of 0.001 and it appears that towards the leading edge, an increase in the shear stress aft of the transition point tends to draw some of the fluid downstream, creating a small valley. As the speed increases and the magnitude of the shear stresses increase, this trend is continued. Meanwhile, prior to rotation, an adverse pressure gradient sets up in the back part of the airfoil and this tends to push the fluid on the trailing edge toward the center of the chord length. Upon rotation, the situation worsens. With the increase in angle of attack, the transition point moves toward the leading edge and the region of increased shear stress moves forward sending the fluid behind it downstream. During this same time, as the angle of attack is increasing, the pressure gradients toward the trailing edge are getting larger. The fluid that is coming from the upstream positions eventually meet the "wall" at the other end due to the larger pressures and the fluid collects near the 80-100% region of the chord. Eventually the problem becomes that of an accumulation of fluid near the trailing edge. The two driving forces go against each other and the solution is not clear.

This process was typical of that observed for the all of the runs attempted. Sometimes, the amount of fluid accumulating near the trailing edge got to be too much and the program stopped as it couldn't handle the large gradients in fluid depth. It is believed that there are limitations in the model equation that was used and that the method of fluid removal is not necessarily one of continuous, smooth motion. In experimental observations, the fluid did accumulate near the trailing edge and break away from the airfoil as the fluid depth became large. This process of one piece of fluid shearing from the other is not accounted for in the model, but there is good agreement between these numerical predictions and experimental observations in the general way in which the fluid accumulates near the trailing edge.

There may be a good physical explanation as to the phenomena that is observed in this problem. In the airfoil, after rotation, the fluid is being pulled to the back of the airfoil by the shear stress and "blocked" by the fluid being pushed to the center by large pressure gradients. In the problem of pancake syrup being poured from a container, gravity sends the syrup toward the pancake and the pancake effectively blocks the fluid (see the artist's rendition of the pancake problem in Figure 6.2). This is similar to the airfoil. We have observed the pancake syrup distributing itself on the pancake in two different ways. When the syrup is hot, the

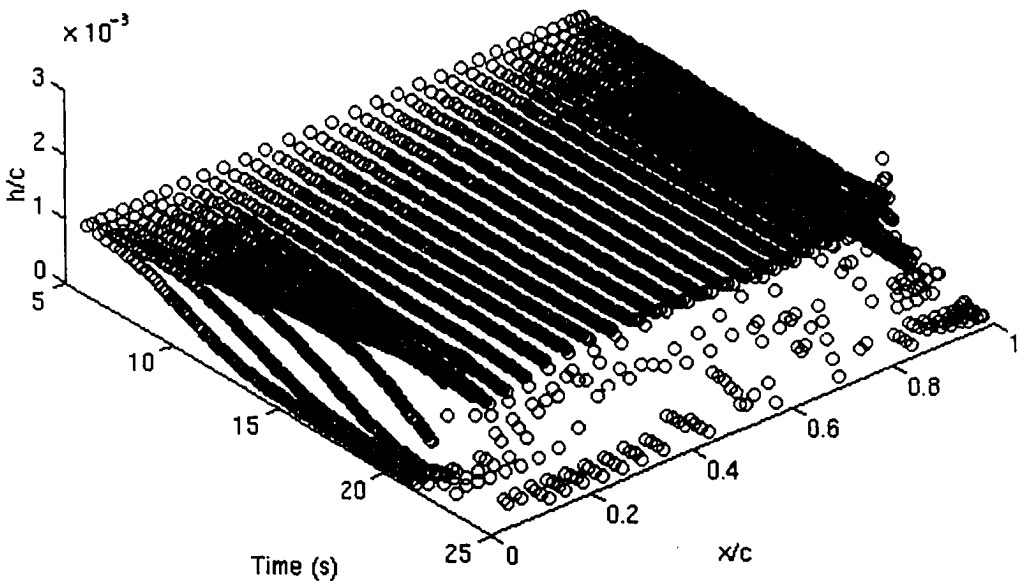


Figure 6.1: Deicing fluid depth as a function of time for one of the tests

viscosity of the fluid is small and the fluid distributes itself evenly over the surface. When the fluid is cold with a higher viscosity, on the other hand, the syrup becomes unstable, buckles, and moves back and forth over the pancake.

What this says is that we may be experiencing the same problem within the deicing fluid. The modeling of the deicing fluid in this study does not take into account fluid stability as might be used in a buckling analysis. It appears that this may be the case as the driving force becomes so large or that the "stopping" force of the pressure gradient induces an instability and a buckling of the fluid. As will be seen with some of the global results, this phenomena appears to be dependent on fluid viscosity. This is perhaps a major conclusion of this work. While the fluid remains stable, the code is justifiable in its conclusions, but once the fluid becomes unstable, the methods of analysis used in this study become inadequate.

6.2.2 Test parameters

Within this study, it was decided that while keeping most of the take-off simulation parameters the same, we might look at the effect of viscosity and initial depth on the global results. There was also one parameter that was changed within the take-off simulation, namely

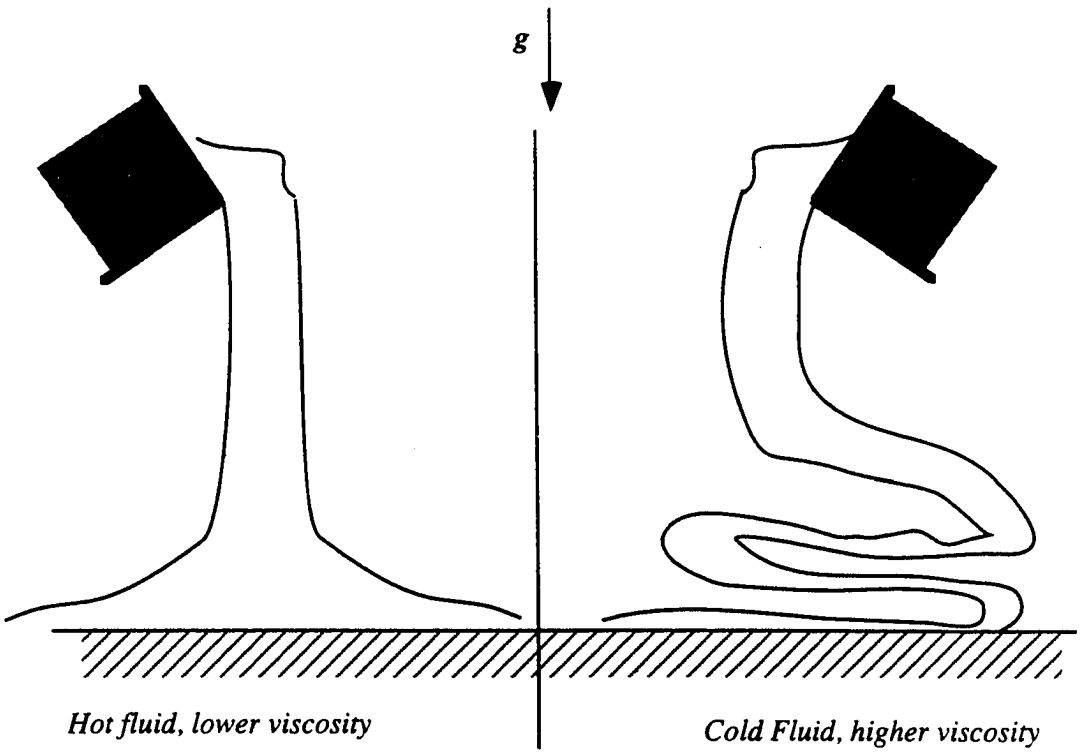


Figure 6.2: The pancake syrup problem

that of rotation speed. So, for an initial angle of attack of 1.0 degree, an initial freestream value of 3.5 m/s and a pitch rate of 3.5 degrees per second after rotation, the effect of changing the rotation speed from 41.0 m/s (the example problem) to 55.0 m/s was included in a test matrix that attempted to test three different ratios of dynamic viscosity and two deicing fluid depths.

Along with dynamic viscosity ratios (between the fluid and air) of 12000, 18000 and 24000, tests were attempted on fluid depths of 0.001 and 0.002 relative to the chord length. One experimental study sited a h_g/c ratio of 0.00125 [10], so these values seem reasonable. The lower rotation speed is expected to simulate the take-off procedure used by a general aviation aircraft, while the larger take-off speeds are more akin to small transport/regional aircraft.

6.2.3 Fluid depth distribution of deicing fluid as a function of viscosity

With this time history in mind, questions naturally arise as to what kinds of fluids exhibit this behavior of backing up toward the trailing edge. Higher viscosity fluid or lower

viscosity fluids? Deeper or shallower fluids? Higher or lower take-off speeds? First of all, we will look at the distribution of at a constant angle of attack to compare the effect of rotation speed. A logical place to do this is at the point at which the time steps were modified by the last condition outlined in Section 6.1. When the angle of attack was at 11 degrees, the distribution of deicing fluid depth for varying viscosities is shown in Figure 6.3 for the shorter take-off run. There are some similarities between the three. All three viscosities have "removed" fluid from the front part of the airfoil by this time. That fluid then collides with the fluid that is backed up by the large pressure gradient in the back of the airfoil.

What this means to global effects is unclear. For this fluid depth, all of the estimates for lift coefficient were within 0.5% of each other and the clean wing data. The last observation is that there appears to be a fair amount of fluid that remains on the wing at this point. The shear-thinning characteristics of the Non-Newtonian fluid may help this, but an estimate of the importance of the non-linearity is beyond the scope of this work.

6.2.4 Fluid depth distribution of deicing fluid as a function of initial fluid depth

For the low-speed take-off, runs were made that tested the effect of initial depth of fluid. As can be seen in Figure 6.4, for the 12000x viscosity fluid, the first at 0.001 h_o/c initial depth and the second one with an initial non-dimensional depth of 0.002, both situations have cleared most of the fluid from the leading edge area by the time the 11 degree angle of attack mark is reached.

The fluid still backs up near the trailing edge, but any modification to the leading edge transitions are probably similar by this point. Again, the global effects of increasing the fluid depth are minimal if non-distinguishable from the clean wing.

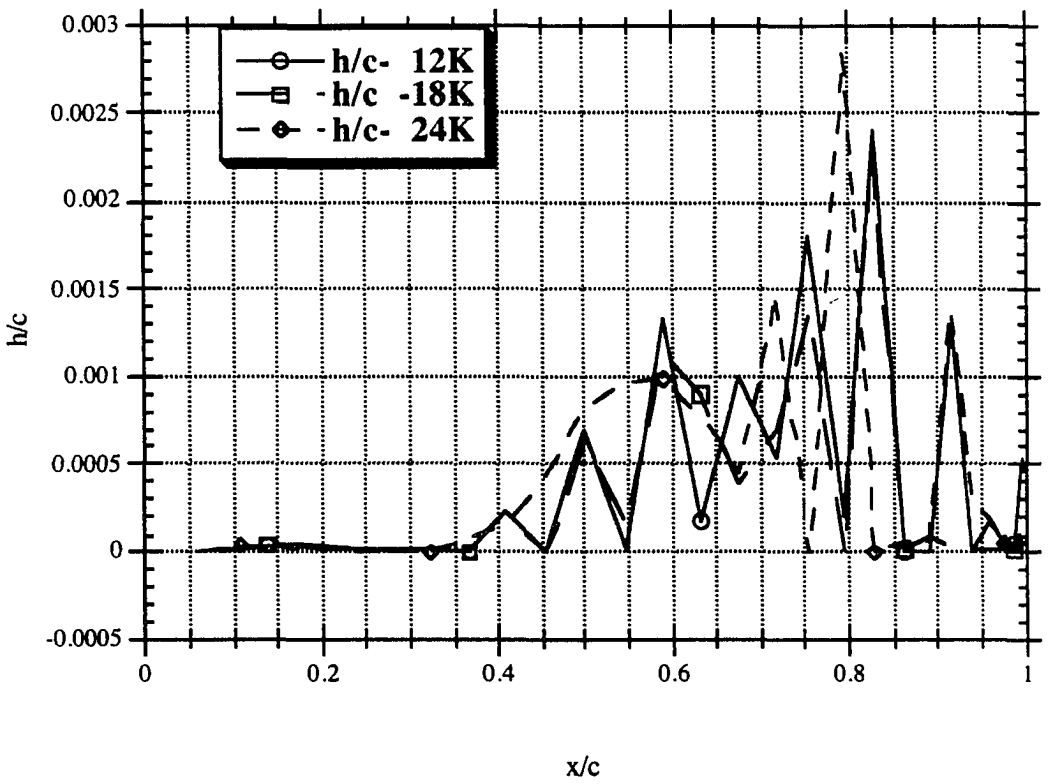


Figure 6.3: Fluid depth for varying viscosity ratios at 11 degrees angle of attack for the short take-off run

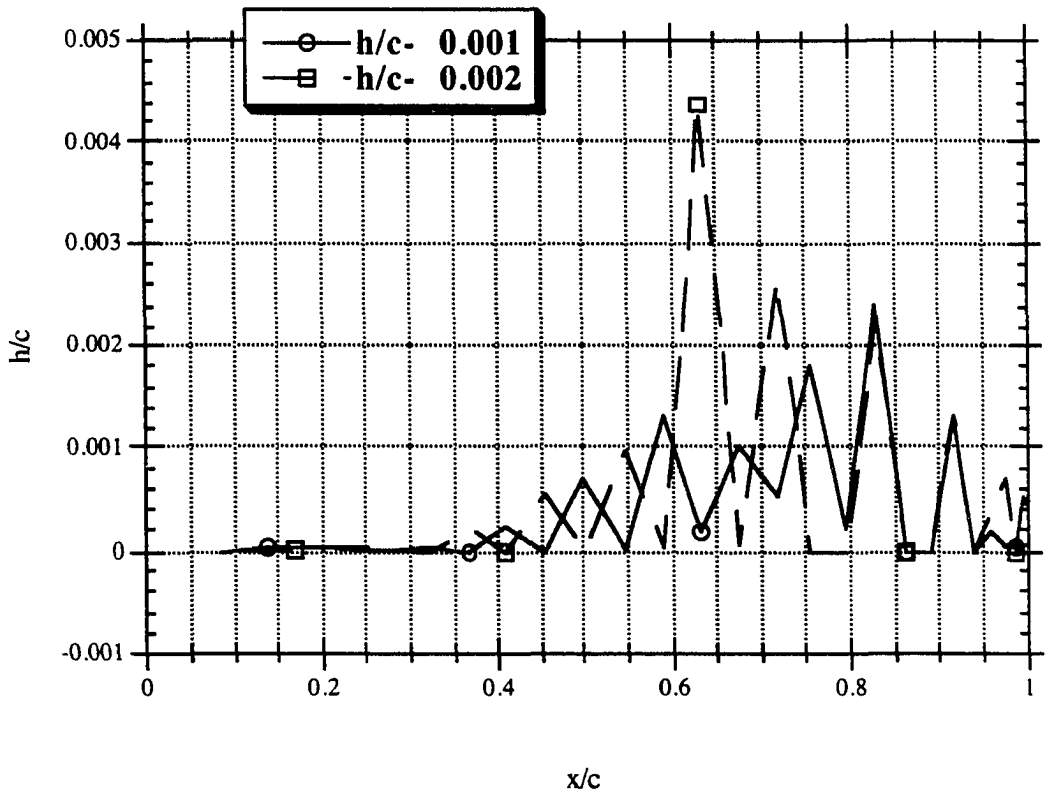


Figure 6.4: Fluid depth distribution for two different initial depths at 11 degrees angle of attack for the short take-off run

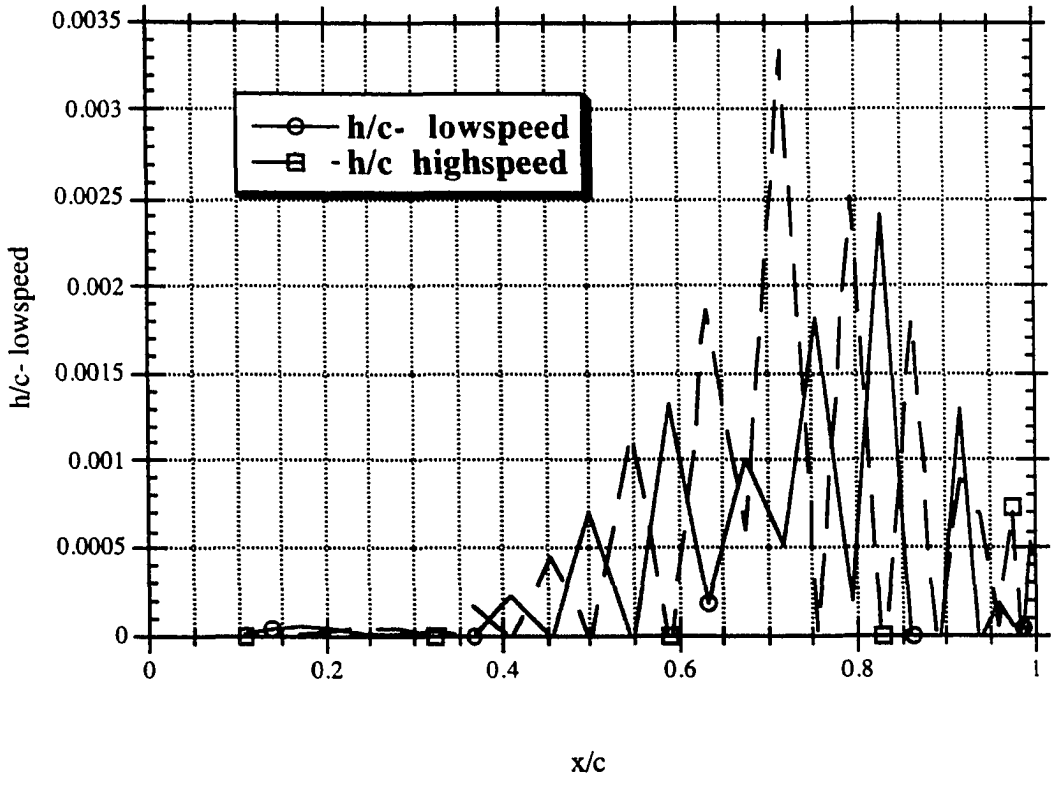


Figure 6.5: Fluid depth distribution for at 11 degrees angle of attack for low speed flow and high speed flow

6.2.5 Fluid depth distribution of deicing fluid as a function of take-off speed

Finally, let us look at how the fluid depth is distributed for a change in rotation speed. At 11 degrees angle of attack, the distribution of fluid depth on the NACA 0012 airfoil is shown in Figure 6.5 for the 12000x viscosity fluid in both cases. Because the rotation speed is so much higher, the pressures at this angle of attack in the "high speed" run (although distributed similarly when normalized to the freestream velocity) are physically much greater (really much more negative) in the suction region. Specifically, the pressure gradient towards the trailing edge is much greater. This would tend to make the fluid near the trailing edge more difficult to remove and we see that it builds up near the trailing edge with more depth than at the

lower speeds. Again, there was minimal effect on the global lift coefficient at this point and ultimately in the maximum lift coefficient.

6.2.6 Final comments on fluid depth distribution

One might surmise that the effects on the transition point to turbulence are minimal as there is not much fluid near the leading edge. Unless the fluid is very viscous, based on the information in these data, if there proves to be a reason as to losses in lift, it is probably not because of modification of the transition point of the gas-dynamic boundary layer. With the small fluid depths, the slip velocity at the interface is almost non-existent and therefore shouldn't be much different than a clean airfoil. This is a major conclusion and an original question of the research.

Secondly, it clear that it may have been useful in this study to distribute more points around the airfoil. As seen in the figures of this section, there are gaps where the fluid is thought to be zero or minimal for "long" periods of length toward the trailing edge of the airfoil. If there are more points to help capture the fluid build-up near the trailing edge, the natural development of the phenomena may be captured better requiring less dependence on the reflection condition described in Chapter 5. Unfortunately, within the limits of practicality, this may not be possible. Running the standard 70 node airfoil takes over 10 hours to complete a full take-off simulation on a dedicated DEC Alpha machine. It appears that the problem may warrant higher performance computers than is available to this author at this time. This is certainly a point to think about for future work.

One final observation is in order. In limited experiments (not detailed here, obviously) it appeared that the fluid would congregate near the trailing edge and then come off in a small burst. This leads me to believe that the fluid build-up will not become infinite in magnitude (See Figure 6.6 where the span of the airfoil is vertical in the figure and the figure shows about 10 cm of span). The assumption of a continuum within the fluid may not be good in this region of operation and another fluid property may be important, perhaps the surface tension in the fluid. In the event that the fluid buildup becomes large in amplitude and short in "wavelength", the second derivatives at the peaks become large and the pressure change across the fluid interface may be different than the assumed zero value. This coupled with the potential buckling problem outlined above leaves the door open for future work in this area.

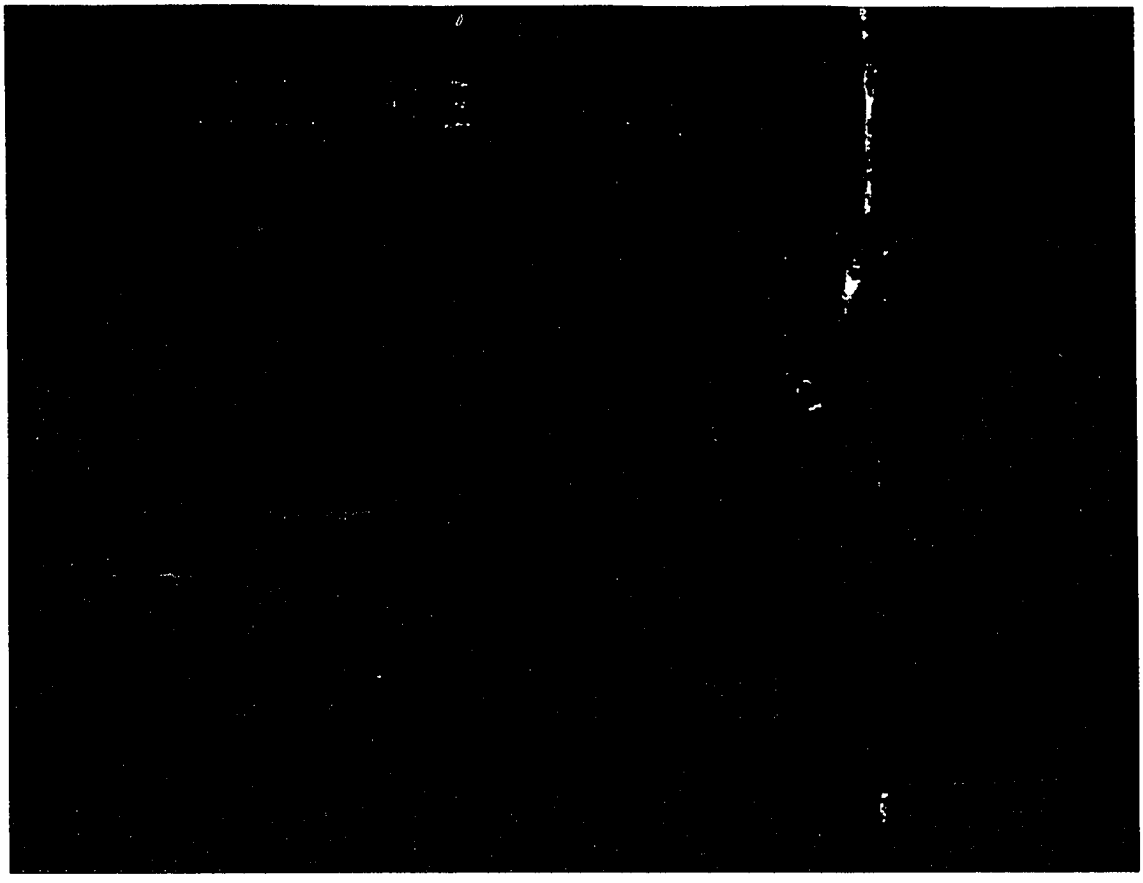


Figure 6.6: Video capture of fluid leaving the trailing edge of an airfoil (TOP VIEW)

6.3 Global results

Finally, we wish to look at the differences in global performance parameters when comparing clean wings and those contaminated by deicing fluid. As outlined earlier, the intended tests were to cover a range of fluid viscosities, initial fluid depths and rotation speeds. Comparing the maximum lift coefficient and separation point appeared to be a good idea at the beginning of this study. There were really only a few cases which we were able to run to completion without the buckling problem arising.

The results for these few tests showed that the inclusion of the deicing fluid in the analysis had very little effect on the maximum lift coefficient (defined as the point at which the

Table 6.1: Global performance results for low-speed tests with low initial fluid depth

	Variable	Run 1	Run 3	Run 4	Run 5
Geometry		NACA 0012	NACA 0012	NACA 0012	NACA 0012
Initial Velocity	V_o	3.5	3.5	3.5	3.5
Rotation velocity(m/s)	V_r	41.0	41.0	41.0	41.0
Acceleration(m^2/s)	dV_∞ / dt	2.5	2.5	2.5	2.5
Initial Angle of Attack(degrees)	α	1.0	1.0	1.0	1.0
Pitch rate(degrees/s)	$d\alpha / dt$	3.5	3.5	3.5	3.5
Initial depth	h_o/c	-	0.0010	0.0010	0.0010
Ratio of viscosities	μ_f / μ_{air}	-	12000	18000	24000
Number of Panels	NN	70	70	70	70
Number of Grid points	NJ	125	125	125	125
$c_{l,max}$		1.5700	1.5580	1.7174	1.4970
α_{max}		15.719	16.060	17.490	15.299
Separation point node		70	69	69	69

flow first experienced separation). Table 6.1 shows the results for the low speed testing for the lowest initial deicing fluid depth.

Increasing the fluid depth for these low speeds presented the problem of potential fluid buckling. The angle of attack at which this occurred is noted in Table 6.2. With the increase in viscosity, the phenomena shows up, even for the low speeds. It is thought that difference in the two points in the 18K and 24K viscosities are not numerically relevant.

Increasing the rotation speed brought the "buckling" phenomena to light even in the lowest depths as shown in Table 6.3. There is a slight difference in the angle of attack at which this program cannot find a solution. With the more viscous fluid, the phenomena occurs slightly earlier.

Upon increasing the rotation speed and the fluid depth, not even the least viscous fluid reached a solution as it experienced difficulties at a maximum lift coefficient of 0.5572 at 4.7015 degrees angle of attack. With this result, no further runs were attempted.

It appears that there is some confidence in the idea that there is a fluid stability problem that is heightened by 1) increased viscosity, 2) increased rotation speeds (larger pressure gradients and shear stresses and 3) increasing amounts of fluid. This is a major conclusion of this work.

It also appears that in this method of analysis, when the fluid does not reach this breakdown, that there is minimal effect on the global performance of the airfoil.

Table 6.2: Global performance results for low-speed tests with high initial fluid depth

	Variable	Run 1	Run 6	Run 7	Run 8
Geometry		NACA 0012	NACA 0012	NACA 0012	NACA 0012
Initial Velocity	V_o	3.5	3.5	3.5	3.5
Rotation velocity(m/s)	V_r	41.0	41.0	41.0	41.0
Acceleration(m^2/s)	dV_∞ / dt	2.5	2.5	2.5	2.5
Initial Angle of Attack(degrees)	α	1.0	1.0	1.0	1.0
Pitch rate(degrees/s)	$d\alpha / dt$	3.5	3.5	3.5	3.5
Initial depth	h_o/c	-	0.0020	0.0020	0.0020
Ratio of viscosities	μ_f / μ_{air}	-	12000	18000	24000
Number of Panels	NN	70	70	70	70
Number of Grid points	NJ	125	125	125	125
$c_{l,max}$		1.5700	1.488	0.2690	0.263
α_{max}		15.719	15.019	2.1499	2.171
Separation point node		70	69	-	-
Speed of buckling(m/s)	V_{max}			41.896	41.908

Table 6.3: Global performance results for high-speed tests with low initial fluid depth

	Variable	Run 2	Run 10	Run 11	Run 12
Geometry		NACA 0012	NACA 0012	NACA 0012	NACA 0012
Initial Velocity	V_o	3.5	3.5	3.5	3.5
Rotation velocity(m/s)	V_r	55.0	55.0	55.0	55.0
Acceleration(m^2/s)	dV_∞ / dt	2.5	2.5	2.5	2.5
Initial Angle of Attack(degrees)	α	1.0	1.0	1.0	1.0
Pitch rate(degrees/s)	$d\alpha / dt$	3.5	3.5	3.5	3.5
Initial depth	h_o/c	-	0.0010	0.0010	0.0010
Ratio of viscosities	μ_f / μ_{air}	-	12000	18000	24000
Number of Panels	NN	70	70	70	70
Number of Grid points	NJ	125	125	125	125
$c_{l,max}$		1.5580	1.5020	0.2640	0.135
α_{max}		16.060	15.239	2.2325	1.015
Separation point node		68	69	-	-
Speed of buckling(m/s)	V_{max}			55.91	55.05

7. RECOMMENDATIONS FOR FURTHER WORK

Since this effort was undertaken with little prior work completed in the field, the expectation is that there remains further study in this important area of flight safety. In this section, the discussion will focus on recommended improvements in the code and further areas for study.

7.1 Limitations of current work and next generation improvements

Within the current study, there are limitations not only in the results, but the methods of analysis. From the RESULTS section of this dissertation, it is clear that not all variables have been explored completely.

Something that the current study is capable of looking into is the effect of airfoil geometry. Only a NACA 0012 airfoil was tested in this preliminary investigation. The anecdotal evidence suggests that certain kinds of airfoil sections are more susceptible to contamination effects than others. The thinner airfoils will produce a larger pressure gradient aft of the suction spike than the thicker, more rounded airfoils for a given angle of attack and pitch rate. There are two leading edge separation phenomena that can occur that will degrade airfoil performance at high angles of attack. Leading edge separation usually occurs in the thicker airfoils where separation happens prior to transition to turbulence. The resulting flowfield will create a separation bubble which can reattach as a "short bubble", in which case the resulting interaction of displacement thickness with the outer flow will give a larger drag component than if the separation bubble was not present. The other option at this point is that of a "long-bubble". This type of bubble does not reattach right away and extends much further down the chord. This is considered catastrophic and difficult to recover from. Either way, leading edge separation is a phenomena which is not indicative of optimal performance. The airfoil geometry is expected to play a large role in the type of bubble formation and is in area for study with future programming.

Numerical methods of leading edge separation prediction are complex and really beyond the scope of this study. One method of modeling the flow within the separation bubble is to assume that the convective term in the boundary layer equations is not present when there exists reversed flow. This is known as the FLARE method [1] and has had reasonable success in predicting the short bubble. The main issue in this method of study is the modeling of

turbulence within the separation bubble. It is generally assumed that upon reattachment, the flow is considered to be turbulent. While this may be true, there is debate on how to model the turbulence within the bubble. For example- prior to separation, the flow is laminar. In the reversed flow region of the separation bubble, at what point are the turbulent shear stresses included? One of the shortfalls of the Prandtl mixing length of turbulence modeling is that it is not terribly good near the separation points (or where the shear stresses are small). It is believed, therefore, that in future work, should one wish to explore the effects of the leading edge separation bubble, that some decision on the point of transition would have to be made and that higher order turbulence models would be in order. For the present study, it is assumed that changes in airfoil geometry will be of interest at a future date, with the resulting effect being the inclusion of information about laminar separation bubbles.

Another recommendation and limitation of the current work involves the take-off simulation. Only two take-off profiles have been studied with the rotation speed being the parameter that changes between the two. There may be some subtle differences in initial angles of attack and pitch rate that qualitatively effect the results. In particular, the effect of pitch rate is very clear in its trend. Regardless of the airfoil geometry, a higher pitch rate manifests itself in a larger adverse pressure gradient aft of the minimum pressure point. Take two pitch rates, one at 3° per second and the second at 4° per second (both are in the range of general aviation maneuvers). The larger pitch rate has the effect of moving the point of minimum pressure away from the stagnation point at a quicker rate. Any fluid particle that begins at the stagnation point that wishes to go around the leading edge to the back of the airfoil must "chase" the leading edge when the angle of attack is increasing, creating a local acceleration. The effect of this higher pitch rate is to increase the maximum value of the suction pressure. This means that aft of the minimum pressure point, if the flow is to recover, it has to go through a larger pressure gradient making the possibility of separation greater. Regardless, this study does not extend to different pitch rates, only looking at the effects of fluid depth and viscosity on a given take-off profile with different rotation speeds. The programming is capable of this distinction, but it has not been explored fully.

So, in the future, different airfoil geometries and take-off profiles may be of interest including the effect of pitch rate after rotation, and the previously unmentioned initial angle of attack and freestream acceleration rates. The values chosen for the initial angle of attack and acceleration rate of the freestream velocity are typical of general aviation aircraft, but there are certainly deviations within aircraft models. Again, the programming is capable of looking at this, but it remains an area for future exploration.

Two obvious areas for further exploration remain. Namely, in this initial study, the effect of the non-linearity of the deicing fluid has been neglected. As discussed in the section dealing with METHODS OF ANALYSIS: THE DEICING FLUID, the inclusion of this effect would require a iterative procedure given a non-linear shear stress-rate of strain relationship within the fluid. In the event this work is used to supplement experimental work on specific fluids, the true characteristics of the fluid should be included.

The last real limitation of this study is the three dimensional effect that has been ignored. It is not clear as to what other insight the inclusion of three dimensional effects would bring. Aside from increasing the amount of computational resources that would be required, the equations of motion would have to be extended to three dimensions. The increase in computational effort is not linear, however. Increasing the study to three dimensions would require orders of magnitude increases in the computational time needed to solve the problem. This author is of the opinion that effort spent on improving the two-dimensional model would be more fruitful than delving into the three dimensional world with the current limitations of the physical models of linear deicing fluid, algebraic turbulence modeling, and limited understanding of the effect of take-off simulation parameters.

Finally, this author admits that he is not particularly pleased with the compromise in time scales that was made in this study as discussed in the RESULTS section. This is a problem that appears to be inherent in the physical problem, though. Realize that these are two different types of flow regimes. The boundary layer analysis is considered to be "high Reynolds number flow" with its approximations in the Boundary Layer equations while the deicing fluid involves "low Reynolds number flow" that involves the neglect of the inertia in the formulation of the governing equation of motion. I believe that the next efforts in this area will have to better model the trailing vortices. Admittedly, the numerical discretization of the analog physical problem is bound to cause some difficulties. An "analog to digital" converter is needed to model the analog distribution in time of the strength of the shed circulation. Perhaps a modification of the Kutta Condition is required such that the stagnation point is allowed to move from the trailing edge. Even though we expect the system to be naturally stable as to move the stagnation point back to the trailing edge, there may be transient effects in the motion of the stagnation point within a short region around the trailing edge- up the top surface, back down to the trailing edge and toward the leading edge on the lower surface until the next "burst" of shed vorticity brings the stagnation point back to the trailing edge and the process begins again.

One other note is that this author suspects that results may be improved by increasing the number of nodes around the airfoil to properly capture the build up of fluid at the trailing edge. Because of practical limitations on the number of nodes (CPU time), a large number of nodes was not tried, but I think the reflection condition used in Chapter 5 would be minimized if there were more nodes to collect data.

8. CONCLUSIONS

The goals of this research effort were fairly ambitious at the outset. Not only did we intend to produce a computer code that would help explain the reported loss in lift during take-off due to application of deicing fluids, but we had hoped to perform wind tunnel testing. Upon closer inspection of the requirements of such a test, the non-linearity of the deicing fluid presented itself and the matter of a distorted model made this option less attractive.

Within this study, three different flow regimes were studied 1) an outer flow, 2) a gas-dynamic boundary layer and 3) a layer of deicing fluid embedded underneath that gas-dynamic boundary layer. The intention was to determine if fluid properties such as high viscosity are particularly detrimental to smaller aircraft that are beginning to use the Type II fluids. The particular case of take-off simulation was studied on a two-dimensional airfoil with a NACA 0012 geometry.

The numerical methods employed to study this phenomena are varied. The outer flow was modeled using a time-dependent PANEL method that shed some interesting light on the variability of pressure distribution on an airfoil in an accelerating flowfield. The modeling of trailing vortices was attempted and results were consistent with expectations. The motion of the leading edge stagnation point with time eventually may prove to be useful in other airfoil contamination problems.

The gas-dynamic boundary layer was modeled using finite difference methods. The time dependence, again, was interesting in how the gridding needed to be modified to follow the flow. With the boundary layer thickness changing in height with time, the grid has to change as well. The local component of acceleration in the pressure gradient reminded us of the time dependence of the problem. Often times, as engineers who have looked at aerodynamics for a "long" time and feel we have a good understanding of the problem, we forget about this vital piece of information. Often times in the development of this code we saw things that didn't make sense at first glance until we recalled the accelerating flowfield or the rotation maneuver.

Transition to turbulence was initially thought to be a major issue upon inclusion of the deicing fluid, but the matching conditions at the interface of the deicing fluid and gas-dynamic boundary layer proved to be inconsequential. Care was taken to keep track of the transition points as defined by criteria based on Reynolds number and shape factor, but for the range of

viscosities checked, the slip velocities were found to be very small compared to freestream velocities. Consistent with experimental observations, any deicing fluid at the leading edge moved away quickly upon the rotation maneuver. This leaves us with the conclusion that any modification to the transition point due to presence of the deicing fluid is probably not responsible for the reported losses in performance.

Within the non-linear deicing fluid governing equation, finite differences were used again to approximate the motion of the fluid given pressure gradient inputs and shear stresses acting at the surface. An iterative process was used to solve the equation of motion maintaining the non-linearity (the deicing fluid depth, h, appears as third order in the equation). The interaction of the deicing fluid with the gas-dynamic boundary layer was implemented in two ways 1) as a modification of the no-slip condition at the interface with the gas-dynamic boundary layer in that a finite slip velocity was introduced into the gas-dynamic boundary layer where the slip velocity was zero before and 2) as a modification of the displacement thickness that interacted with the outer flow to modify the outer pressure distribution. The conclusion reached is that the effect of the no-slip condition on clean wing solutions is minimal in comparison to the displacement thickness effects. Adding a displacement thickness equal to the depth of fluid produced an effectively thicker airfoil and thus asks for a larger convective acceleration and results in a slightly higher lift coefficient than the clean wing. This was as expected.

As for the final question, "Does the addition of deicing fluid result in performance loss?", the answer is: "The results of study do not indicate this is so." For the situations of dynamic viscosity ratios, take-off simulation parameters, and initial fluid depth, there were no noticeable differences in estimates of maximum section lift coefficient or the lift curve slope. However, an important realization was made even with the deicing fluid assumed to be linear in its shear stress-rate of strain relationship.

As the fluid is pulled to the back of the airfoil by the shear stresses acting at the interface of the fluid, its comes up against fluid that is being pushed from the rear by the increasing pressures. This can continue for a while, especially at low speeds or low angles of attack, but eventually upon the rotation, both of these kinds of effects are increased. The values for the shear stress as well as the short favorable pressure near the leading edge become larger and the pressure gradient at the trailing edge becomes more adverse. With these two counter-acting forces becoming more at-odds, the assumptions used in this study break down and the physical explanation for the poor solutions is that there is a fluid buckling at the region near the trailing edge where the fluid is accumulating. The modeling in this study is not

sophisticated enough to handle this flow situation and the solution becomes intolerable. Lending confidence to this conclusion is the trends seen in the point at which the solution deteriorates.

The results show that the solution breaks down under 1) increased rotation speed, 2) deeper initial depths and 3) increasing fluid viscosity. Without the proper model in place, there is no hope of capturing this phenomena.

A word is in order on the practicality of these results and of this study. There is a thought process that goes as follows. In a high-speed rotation typical of larger aircraft, even if the fluid viscosity is large, once the fluid begins to accumulate near the trailing edge, it is ripped away from its neighbor and flung from the airfoil by the shear stresses leaving less fluid near the trailing edge. This was seen in limited experiments that we have done. The fluid does not flow smoothly, but rather in discrete bursts. The fluid then accumulates again and is separated from the surface by the relatively large shear stresses. This process continues and eventually, the surface is nearly free of residual fluid.

In a smaller aircraft, the fluid still accumulates, but the shear stresses are not large enough to overcome the fluid remaining attached to the surface until the depth becomes larger than before. Because of the requirement that the fluid depth become large, the process of discrete shearing does not take place as often as in the aircraft with the larger rotation speed. This leaves residual fluid and the effect is a an displacement thickness which can lead to premature separation.

What remains to fully understand this phenomena is a study of the fluid buckling effects near the trailing edge and to incorporate the non-linearity into the analysis. I don't believe it is the non-linearity that causes the problems in performance, but rather the increased relative viscosity of the fluid. The fluid buckling may not provide all the answers as the mechanism by which the deicing fluid is separated from the airfoil is still unknown.

WORKS CONSULTED

[1] Anderson, D.A., Tannehill, J.C., and Pletcher, R.H. Computational Fluid Mechanics and Heat Transfer. New York: Hemisphere Publishing Corporation, 1984.

[2] Berkowitz, Brian, et al. "Experimental Ice Shape and Performance Characteristics for a Multi-Element Airfoil in the NASA Lewis Icing Research Tunnel." NASA Technical Memorandum 105380: 1-9.

[3] Bragg, M.B. "Experimental Aerodynamic Characteristics of NACA 0012 Airfoil with Simulated Glaze Ice." Journal of Aircraft. Vol. 25, No. 9, (1988): 849-854.

[4] Bragg, M.B., and Coirier, W.J. "Aerodynamic Measurements of an Airfoil with Simulated Glaze Ice." AIAA 86-0484, 24th Aerospace Sciences Meeting and Exhibit, 6-9 January 1986, Reno, NV: 1-11.

[5] Bragg, M.B., and Coirier, W.J. "Detailed Measurements of the Flowfield in the Vacinity of an Airfoil with Glaze Ice." AIAA 85-0409, 23rd Aerospace Sciences Meeting and Exhibit, January 14-17, 1985, Reno, NV: 1-10.

[6] Bragg, M.B., and Khodadoust, A. "Measurements in a Leading-Edge Separation Bubble due to a Simulated Airfoil Ice Accretion." AIAA Journal. Vol. 30, No. 6 (1992): 1462-1467.

[7] Bragg, M.B., et al. "Airfoil Aerodynamics in Icing Conditions." Journal of Aircraft. Vol. 23, No. 1, (1986): 76-81.

[8] Bragg, M.B., Kerho, M.F. and Khodadoust, A. "LVD flowfield measurements on a straight and swept wing with a simulated ice accretion." AIAA 93-0300, 31st Aerospace Sciences Meeting and Exhibit, January 11-14, 1993, Reno, NV: 1-19.

[9] Charles, M.E., and Lilleleht, L.U. "An experimental investigation of stability and interfacial waves in co-current flow of two liquids." Journal of Fluid Mechanics. Vol. 22, part 2, (1965): 217-224.

[10] Ellis, N., Lim, E., Teeling, P. and Zhu, S. "Wind Tunnel Tests of Aerodynamic Effects of Type I & II Ground De/Anti-icing Fluids on Small Transport & General Aviation Aircraft during Takeoff." AIAA 91-0763, 29th Aerospace Sciences Meeting and Exhibit, January 7-10, 1991, Reno, NV: 1-14.

[11] Goldstein S., and Rosenhead, L. "Boundary layer growth." Proc. Cambr. Phil. Soc. No. 32, (1936): 392-401.

[12] Griffiths, Robert C. "Investigation of Leading Edge Ice Accretion With Cyclical Pneumatic Boot Inflation." AIAA 93-0007, 31st Aerospace Sciences Meeting and Exhibit, January 11-14, 1993, Reno, NV: 1-13.

[13] Hassan, H.A. "On unsteady boundary layers." Journal of Fluid Mechanics. Vol. 9, (1960): 300-304.

[14] Hedde, T., and Guffond, D. "Improvement of the ONERA 3D Icing Code, Comparison with 3D Experimental Shapes." AIAA 93-0169, 31st Aerospace Sciences Meeting and Exhibit, January 11-14, 1993, Reno, NV: 1-10.

[15] Hess, J.L., and Smith, A.M.O., "Calculations of Potential Flow About Arbitrary Bodies." Progress in Aeronautical Sciences. Vol. 8, (1966): 1-139.

[16] Hill, Eugene G., and Zeirten, Thomas A. "Flight and Wind Tunnel Tests of the Aerodynamic Effects of Aircraft Ground Deicing/Anti-Icing Fluids." AIAA 91-0762, 29th Aerospace Sciences Meeting and Exhibit, January 7-10, 1991, Reno, NV: 1-14.

[17] Hooper, A.P., and Boyd, W.G.C. " Shear-flow instability at the interface between two viscous fluids." Journal of Fluid Mechanics. Vol. 128, (1983): 507-528.

[18] Hornig, R. "Development of an International Standard for Safe Winter Operation." Journal of Aircraft. Vol. 30, No. 1, (1993): 14-18.

[19] Hughes, David. "European-Developed Anti-icing Fluid Expected to Gain Acceptance in U.S." Aviation Week and Space Technology. March 30, 1992: 29.

[20] Joseph, D.D. Stability of Fluid Motions I and II. Vol. 27 and 28 of Springer Tracts in Natural Philosophy. Berlin: Springer-Verlag, 1976.

[21] King, A.C., Tuck, E.O., and Vanden-Broek, J.-M. "Air-blown waves on thin viscous sheets." Physics of Fluids, A5, April 1993: 973-978.

[22] Kinney, R.B., and Cielak, Z.M. "Analysis of unsteady viscous flow past an airfoil, Part II: Numerical formulation and results." AIAA Journal. Vol. 16, (1978): 105-110.

[23] Kinney, R.B., and Cielak, Z.M. "Analysis of unsteady viscous flow past an airfoil, Part I: Theoretical developments." AIAA Journal. Vol. 15, (1977): 1712-1717.

[24] Manningham, Dan. "Ice Capades." Business and Commercial Aviation. November, 1986: 67-71.

[25] Manningham, Dan. "Keeping Current: An Icing Review." Aviation Week and Space Technology. April 6, 1992: 32.

[26] McKenna, James T. "New Deicing Procedures: The Push is On." Aviation Week and Space Technology. June 29, 1992.

[27] Meersman, Tom. "New Rules for airport de-icers: Amount of chemicals flushed into river will be reduced." Minneapolis Star-Tribune, September 29, 1993: 1B.

[28] Mehta, V.B., and Lavan, Z. "Starting vortex, separation bubbles and stall: A numerical study of laminar unsteady flow around an airfoil." Journal of Fluid Mechanics. Vol. 16, (1975): 227-256.

[29] Moran, Jack. An Introduction to Theoretical and Computational Mechanics. New York: John Wiley and Sons, 1984.

[30] Ozdemir, I.B., and Whitelaw, J.H. " An optical method for the measurement of unsteady film thickness." Experiments in Fluids. No. 13, (1992): 321-331.

[31] Potapczuk, Mark G., et al. "Ice Accretion and Performance Degradation Calculations with NASA LEWICE/NS." AIAA 93-0173, 31st Aerospace Sciences Meeting and Exhibit, January 11-14, 1993, Reno, NV: 1-20.

[32] Reinmann, John J. "Aerodynamic Effects of Deicing and Anti-Icing Fluids." Journal of Aircraft. Vol. 30, No. 1 (1993): 8-9.

[33] Ross, F. "The contrasting requirements for Type II de/anti-icing fluids." AIAA 91-0759, 29th Aerospace Sciences Meeting and Exhibit, January 7-10, 1991, Reno, NV: 1-5.

[34] Salvador, Rene. "Application and specification of de/anti-icing fluids for aircraft with different short take off time and rotation speed." AIAA 91-0560, 29th Aerospace Sciences Meeting and Exhibit, January 7-10, 1991, Reno, NV: 1-9.

[35] Sankar, L.N., Phaengsook, N., and Bangalore, A. "Effects of Icing on the Aerodynamic Performance of High Lift Airfoils." AIAA 93-0026, 31st Aerospace Sciences Meeting and Exhibit, January 11-14, 1993, Reno, NV: 1-19.

[36] Shin, Jaiwon, et al. "Prediction of Ice Shapes and Their Effect on Airfoil Performance." AIAA 91-0264, 29th Aerospace Sciences Meeting and Exhibit, January 7-10, 1991, Reno, NV: 1-21.

[37] Society of Automotive Engineers. "Aircraft deicing/anti-icing methods with fluids, for large transport aircraft." SAE Aerospace Recommended Practice ARP4737, (1992).

[38] Society of Automotive Engineers. "Fluid, Aircraft Deicing/Anti-icing, Non-Newtonian, Pseudo-plastic, SAE Type II." SAE Aerospace Material Specification AMS 1428, (1993).

[39] Telionis, D.P., and Tsahalis, D.Th. "Unsteady turbulent boundary layers and separation." AIAA Journal. Vol. 14, (1976): 468-474.

[40] van Hengst, J. "Aerodynamic Effects of Ground De/Anti-Icing Fluids on Fokker 50 and Fokker 100." Journal of Aircraft. Vol. 30, No. 1, (1993): 35-40.

[41] White, Frank M. Viscous Fluid Flow, 2nd Edition. New York: McGraw-Hill, Inc., 1974.

[42] Yih, Chia-Shun. "Instability due to viscosity stratification." Journal of Fluid Mechanics. Vol. 27, part 2, (1967): 337-352.

[43] Yih, Chia-Shun. "Wave formation on a liquid layer for de-icing airplane wings." Journal of Fluid Mechanics. Vol. 212, (1990): 41-53.

ACKNOWLEDGMENTS

In this work, I have many people who have been instrumental in making this dream happen. First of all, Drs. Bruce Munson and Jerry Vogel have both provided guidance, advice, compassion and experience to a project that from the beginning, was difficult.

Working an unfunded problem and one that required much attention to the problem formulation in starting from scratch, I hope that each of them enjoyed the experience of working with each other, for they are both truly wonderful people.

There are many things I learned from them; specifically from Bruce, the patience required to be a good teacher. This may never show up in a dissertation, but I hope that in the future, I will be able to use some of these skills that Bruce has taught me to become half the educator he is. Bruce's own patience with me as I traveled the road to this degree was phenomenal. I appreciate Bruce's effort in helping write this dissertation and would hope that in the future we can work together regardless of where I am. I hope that this project renewed a little glimmer in him that reminded him that research can bring rewards of personal satisfaction just for the sake of knowing about something. I can see his compassion in the way he has handled my situation. I will probably come to appreciate his outlook even more as the years separate this work from my thoughts. I also give credit to Bruce for the pancake syrup problem. Wow.

From Jerry, I learned the value of a practical problem. I came to him through Bruce and we spent quite a bit of time determining a suitable problem. I don't think that when we started the problem, we could foresee the path that lead us to this ending point. Through Jerry, I learned the value of grantsmanship and the art of getting others to fund your research. I suspect that in the future, this will be valuable as well. Jerry has instilled in me a need to keep my feet on the ground when tackling research. Through Jerry, I met the people at AERS/Midwest for whom I worked for a short period prior to the completion of this work. For better or worse, the experiences at AERS/Midwest have been interesting, to say the least, and I hope that they will help in carrying me to the next step in my career.

I have one more very important person to thank at this point. My wife, Beth, has been the most understanding, loving, hard-working person I have ever met in my life. Words cannot describe the feelings I have about her giving her time to make our marriage a successful one. Her ability to complete her own degree here at Iowa State in the time I have taken has

been truly incredible. Being apart for the time I was with AERS/Midwest was something that no couple should have to go through. During this time, Beth remained in Ames and kept the house going essentially by herself. I cannot imagine the reserves she had to call on to allow me to make the efforts that I thought would be best for our family. She is an incredible person with whom I look forward to spending the rest of my life with. Thank you, sweetheart. I never would have done this without you!

A few other bits of formal recognition are deserved. Dave Holger, our recently departed department chair deserves attention for his decisions to allow me to teach a variety of courses. I know this will be helpful in pursuing my long-term goals. He didn't have to let me teach all the different courses, knowing it would slow my graduation, but he is truly an educator. He recognized, I think, an enthusiasm on my part to teach and he nurtured it at every opportunity.

The Boeing Company deserves a note of appreciation as they have supported my efforts to complete my degree through a graduate dissertation fellowship. I only hope that in the long run, I can prove their choice wise by becoming a quality engineer.

To the other members of my committee: Ambar Mitra, Jim Iversen and Mike Simonson, I thank you, too, for caring enough to serve on my committee. It takes a group of people to get things done around here, and you are all part of the process that made me the engineer I am today. The timing of these last few weeks left them in a difficult position regarding a thorough evaluation of my work. I realize I pushed the limits of what was acceptable in terms of getting them a copy of my dissertation in a timely manner prior to the final oral exam. This should be no reflection on Jerry or Bruce. I have stretched these limits myself in order to have a graduation day this fall in which my wife and I can both participate. The difference between graduating now or two weeks from now would make no difference in my career pursuits, but in allowing me these extras at the end, you have given me something that Beth and I will remember for the rest of our lives. Thank you.

Throughout my graduate career, I've had a bunch of people that have made this whole trip enjoyable. Dan Kruger, Matt Carney, Mike Sellberg and Joel Ness, currently or formerly of the AE EM department, have been there through most of it. Either as colleagues or teammates, they have provided camaraderie and friendship. I only hope that what I had to offer was enough. Other friends from within the department have been there to bounce ideas off of, commiserate with, and generally share the experience. Shiva Shenoy my long-time office-mate and friend, Rick Hale, Rick Zrostlik, Janice Pawlowski, Rajesh Bashkaran, Brian Frake, Wade Heubsch, Chad Bouton, Tad Calkins, Doug Bradley, Jason Gillette, Christoph

Heimecke, Dan Black, Jane Johnson, Mark Knoerzer, Bijoyendra Nath, Paul Tsao, and Laith Zori, are all people I will be glad to count as colleagues when this is over.

In the department, Don Young, Loren Zachary, Tom Rudolphi, Tom Rogge, Alison Flatau, Len Wilson, Rich Hindman, Dan Adams, Vinay Dayal, P.J. Hermann, Jeff Huston, George Inger, Ken McConnell, Ganesh Rajagopalan, Leroy Sturges, Alric Rothmayer, Adin Mann, Frank Rizzo, and John Tannehill as well as Ted Okiishi from the Mechanical Engineering Department have either been instrumental in developing my teaching skills or my knowledge base. Years from now, when I read this, I don't want to forget any of these people. Thank you.

Our secretaries, Gayle Fay, Tammy Boyd, and Sally Evans have always been helpful. I thank them for their patience with me. Bill Jensen, our wind tunnel technician was helpful in completing work after hours. Thank you. Jim Wellman kept my account active during my stay at AERS and found an extra 10 MB of disk space for me to use these last couple of weeks. Thanks, Jim.

I have two wonderful sets of parents now. My own parents, Patrick and Frances, deserve appreciation for supporting me in my quest. Even when everyone was wondering- "When is Denny going to get a real job?", they stood by me and my decisions. Their support when things were difficult was more than I deserved. My newest set of parents, Del and Karen Eilers, have also been there for us over the last several years. I don't know what it is like to have a daughter and see her marry someone like me, but through this process of graduate school, they have been very supportive of us and our decisions. The love of both of my sets of parents has been something I know I can count on.

My only brother, Michael, and I have become closer over the last couple of years and I want him to know I appreciate his being my "Twin Cities contact". He's been- well- just like a brother. I would do just about anything for him and I know he is going to be successful in whatever he chooses to do.

My grandmother, Yvette Leger, had the foresight to give me the money for my current computer. I am sure I couldn't have done this work without it. There are many things I would like to thank her for, but this was very special. Thank you, Grandma. My other grandma, Kathryn Cronin left us in the summer of '93. I miss her, too, and know she would be proud. I hope that just because my family is "out of sight", they don't think they are "out of mind". I think about them often and love them very much.

My new family, Kathy and Meg Eilers and Kathy's husband, Scott Smallige, have made the trips home really great. I'm really glad I have two sisters now and I really hope that

they get what they want out of life. They have made me feel welcome in my new family. Thank you.

Just a few more people. Rick Wemark, Mike Owen, Tom Plowman, Todd Petersen, Mike Simonson (again), Sam Houk, Luther Schmidt, John Morris, Craig Weiss, John Hinz, Todd Frank, Keith Harmoney, Mike Mallas, Matt Judge, and Duane Wolf have been teammates and friends that have made this trip through graduate school enjoyable. Thanks, guys!

The last group goes without comment, but at some point in my education, they were there to make a difference: Mike Semling, Linda Carlson, Bill Trenda, Fred Adkins, Dave Legrid, Jerry Manthey, Ken Sundstrom, John Gannon, Dave Rykken, Betty Axtelle, Jerry Lee, Tom Yelle, Pat Plant, Bob Pierce, Abe Berman, Bill Gerrard, Brad Liebst, Jack Moran, Ted Wilson, Dale Enns, Tesfahun Berhe, John Reed, John Manush, Mike Lamb, Rob Matthews, Ron Helm, John Monson, Paul Baude, Dave and Julie Bjostad, Dave Kuennen, Justin Windham, Dominic Kessler, Manuel Grados, Dave and Jeff Barton, Todd Sanger, John Butwinick and the people from The Scoresheet Mailing List. Thank you.

APPENDIX A: PROGRAM LISTING

```
c-------------------------------------------------------------
c       djcronin@iastate.edu
c
c       Copyright © Dennis James Cronin, 1995
c       All rights reserved.
c
c       December,1995
c
c       This is the total program.
c-------------------------------------------------------------
        program driver
        dimension r(4000000)
        character *40, title
        real B, eta, muratio
        iyfflag = 0
        ifinal = 0
        B = 0.0
        t = 0.0
        IGF = 0
        sor = 0.30
        itsover = 0
        kgood = 0
        itertimemax=12
        isteady = 0
        markthrough = 0
        tsteady = 0.0
        iseplamt = 0
        iseplamb = 0
        isept = 0
        isepb = 0
        ineg = 0
        icrap = 0
c-------------------------------------------------------------
c       IGF is a flag that goes with geometry
c       isteady is for steady flow in the potential flow solution
c-------------------------------------------------------------
c-------------------------------------------------------------
c       get the airfoil geometry.  either create it, or read from a file
c-------------------------------------------------------------
 10     print 101
        read *, n
        if (n .eq. 3) then
             go to 1000
        endif
        if (n .ne. 1 .and. n .ne. 2) go to 10
        if (n .eq. 2) go to 20
        if (n .eq. 1) then
             call geominput(title,nu,nl,nn,nacanumber,IGF,AR)
             call getnn(nu,nl,nn,title,IGF)
```

```
            i1 = 1
            i2 = i1 + (nn+1)
            i3 = i2 + (nn+1)
            if (IGF .eq. 1) then
            call grid(nacanumber,nu,nl,nn,r(i1),r(i2),title,IGF)
            elseif (IGF .eq. 2) then
            call cylinder(nl,nu,nn,r(i1),r(i2),AR,title,IGF)
            endif
        endif
 20     call getnn(nu,nl,nn,title,IGF)
        i1 = 1
        i2 = i1 + (nn+1)
        i3 = i2 + (nn+1)
 30     call reader(nu,nl,nn,title, r(i1), r(i2),IGF)
c-----------------------------------------------------------------
c       get the other input paramters
c-----------------------------------------------------------------
 40     print 102
        read *, n
        if (n .ne. 1 .and. n .ne. 2) go to 40
        if (n .eq. 2) go to 60
        if (n .eq. 1) then
             call createinput(nj,yf,deltat,tfinal,ratio)
        endif
 60     call getinputparameters(nj,yf,deltat,tfinal,ratio)
        go to 140
 140    print 103
        read *, n
        if (n .ne. 1 .and. n .ne. 2) go to 140
        if (n .eq. 2) go to 161
        if (n .eq. 1) then
             call control(ho, hxbyc, alpha0, alphadot, vinf0,
    +            vrotate, vinfdot, muratio,ivisc,inorm)
        endif
 161    call getcontrol(ho, hxbyc, alpha0, alphadot, vinf0,
    +   vrotate, vinfdot, muratio,ivisc,inorm)
        go to 1010

 1000   call getnn(nu,nl,nn,title,IGF)
        i1 = 1
        i2 = i1 + (nn+1)
        i3 = i2 + (nn+1)
        call reader(nu,nl,nn,title, r(i1), r(i2),IGF)
        call getinputparameters(nj,yf,deltat,tfinal,ratio)
        call getcontrol(ho, hxbyc, alpha0, alphadot, vinf0,
    +   vrotate, vinfdot, muratio,ivisc,inorm)
 1010   nt = 400
        vtest = vinf0
 1011   if (vtest .ge. vrotate) then
        go to 1020
        endif
        nt = nt+1
        deltattest = 1./vtest
        vtest = vtest + deltattest*vinfdot
        go to 1011
```

```
c-----------------------------------------------------------------------
c       Set the pointers
c       ----------------
c       in file 'pointers'
c-----------------------------------------------------------------------
 1020   print *, 'Number of time steps: ', nt
        i4 = i3 + nn
        i5 = i4 + nn
        i6 = i5 + nn
        i7 = i6 + nn
        i8 = i7 + (nn * (nn+1))
        i9 = i8 + ((nn+1)*(nn+1))
        i10 = i9 + (nn + 1)
        i11 = i10 + (nn + 1)
        i12 = i11 + (nn + 1)
        i13 = i12 + nn
        i14 = i13 + nn
        i15 = i14 + (nn*nn)
        i16 = i15 + nn + 1
        i17 = i16 + nn
        i18 = i17 + nn
        i19 = i18 + nn
        i20 = i19 + nn
        i21 = i20 + ((nn+1)*(nn+1))
        i22 = i21 + nt
        i23 = i22 + nt*nt
        i24 = i23 + nt*nt

        i25 = i24 + ((nn+1)*nj)
        i26 = i25 + ((nn+1)*nj)
        i27 = i26 + ((nn+1)*nj*2)
        i28 = i27 + ((nn+1)*nj*2)
        i29 = i28 + ((nn+1)*2)
        i30 = i29 + (nj)
        i31 = i30 + (nj)
        i32 = i31 + (nj)
        i33 = i32 + (nj)
        i34 = i33 + ((nn+1)*2)
        i35 = i34 + ((nn+1)*2)
        i36 = i35 + ((nn+1)*2)
        i37 = i36 + ((nn+1)*2)
        i38 = i37 + ((nn+1)*2)
        i39 = i38 + nj
        i40 = i39 + nj
        i41 = i40 + nj
        i42 = i41 + nj
        i43 = i42 + ((nn+1)*2)
        i44 = i43 + (nn+1)
        i45 = i44 + (nn+1)
        i46 = i45 + (nn+1)
        i47 = i46 + (nn+1)
        i48 = i47 + 4
        i49 = i48 + 4
        i50 = i49 + 3
        i51 = i50 + ((nn+1)*nj)
```

```
      i52 = i51 + ((nn+1)*nj)
      i53 = i52 + ((nn+1)*nj)
      i54 = i53 + ((nn+1)*nj)
      i55 = i54 + ((nn+1)*nj)
      i56 = i55 + ((nn+1)*nj)
      i57 = i56 + ((nn+1)*nj)
      i58 = i57 + ((nn+1)*nj)
      i59 = i58 + ((nn+1)*nj)
      i60 = i59 + ((nn+1)*nj)
      i61 = i60 + ((nn+1)*nj)
      i62 = i61 + ((nn+1)*nj)
      i63 = i62 + ((nn+1)*nj)
      i64 = i63 + ((nn+1)*nj)
      i65 = i64 + ((nn+1)*nj)
      i66 = i65 + ((nn+1)*nj)
      i67 = i66 + ((nn+1)*nj)
      i68 = i67 + (nn+1)
      i69 = i68 + (nn+1)
      i70 = i69 + ((nn+1)*nj)
      i71 = i70 + ((nn+1)*nj)
      i72 = i71 + ((nn+1)*nj)
      i73 = i72 + nn
      i74 = i73 + nn
      i75 = i74 + nn
      i76 = i75 + nn
      i77 = i76 + ((nn+1) * 2)
      i78 = i77 + nn
      i79 = i78 + nn
      i80 = i79 + ((nn+1) * 2)
      i81 = i80 + nn

      mem = 4000000 - i81
      Print *, 'Number of memory allocations left: ',     mem
      if (mem   .le. 0)     then
      print *, 'Too much data.'
      go to 10
      endif

      open(unit = 8, file = 'coef.dat',status = 'unknown')
      open(unit = 72, file = 'totalh.dat',status = 'unknown')
      open(unit = 73, file = 'totaltautop.dat',status = 'unknown')
      open(unit = 74, file = 'totaldeltatop.dat',status = 'unknown')
c     open(unit = 75, file = 'totaldelta2top.dat',status = 'unknown')
c     open(unit = 76, file = 'totaldelta3top.dat',status = 'unknown')
c     open(unit = 77, file = 'totalcptop.dat',status = 'unknown')
c     open(unit = 83, file = 'totaltaubot.dat',status = 'unknown')
c     open(unit = 84, file = 'totaldeltabot.dat',status = 'unknown')
c     open(unit = 85, file = 'totaldelta2bot.dat',status = 'unknown')
c     open(unit = 86, file = 'totaldelta3bot.dat',status = 'unknown')
c     open(unit = 87, file = 'totalcpbot.dat',status = 'unknown')

      write (8,750)
c-----------------------------------------------------------------
c     from the potential flow code, calculate the coefficient matrix,a
c
```

```
c       call coef(xbar,ybar,x,y,rlen,theta,r,beta,a,nn,eta)
c-----------------------------------------------------------------
        call coef(r(i3),r(i4),r(il),r(i2),r(i5),r(i6),r(i7),r(il4),
     +  r(i8),nn,eta)
c-----------------------------------------------------------------
c       now, the time dependent stuff
c-----------------------------------------------------------------
        yftop = yf
        yfbottom = yf

        call freestream(vinf,t,0,tsteady,vinf0,vrotate,
     +                  vinfdot,trotate)
        call start(nn,vinf,r(i72),r(i3),inorm)
        deltat = 2.00*(1./vinf)
        print *, 'Begining deltat:', deltat
 50     k=1
        t = 0.0
        IGF = 0
        isteady = 0
        iyfflag = 0
        normalflag = 0
        tsteady = 0.0
        ksteady = 0
        iturb = 999
c       itransflag = 999

        do 100 k = 1, nt-1
        markthrough = 0
        itertime = 0
 170    t = t+deltat
        call freestream(vinf,t,isteady,tsteady,vinf0,vrotate,
     +                  vinfdot,trotate)
        call angleofattack(attack,alpha0,alphadot,t,trotate)
        print *, ' '
        print *, 'Time:',t
        Print *, 'deltat:',deltat
        print *, 'Freestream velocity:', vinf
        print *, 'Angle of attack:', attack
        if (k-ksteady .eq. 1) then
        call freestream(v0,t,1,tsteady,vinf0,vrotate,
     +                  vinfdot,trotate)
        endif
        if (k-ksteady .ne. 1) then
        call freestream(v0,t-deltat,isteady,tsteady,vinf0,vrotate,
     +                  vinfdot,trotate)
        endif
c-----------------------------------------------------------------
c       this is for the potential flow at any time, t
c       call changev(vinf,v0,vnorm,nn)
c
c       this depends on if 'start' is called.
c-----------------------------------------------------------------
        call changev(vinf,v0,r(i72),nn)
c-----------------------------------------------------------------
c       soln(nn, vinf, attack, b, theta, a, bigwork, ipvt,
```

```
c     +       work, q,rlen,k,circ,circx,circy,nt,xbar,ybar,
c     +       deltat,vnorm)
c-----------------------------------------------------------------
 160     call soln(nn, vinf, attack, r(i9), r(i6), r(i8), r(i20), r(i10),
     +       r(i11), r(i15),r(i5),k,r(i21),r(i22),r(i23),nt,r(i3),
     +       r(i4),deltat,r(i72),vrotate,kgood)
c-----------------------------------------------------------------
c       call potentials(nn, q, xbar, ybar, rlen, phi,
c     +  phis, phiv, theta, x, y,attack,vinf,phiold,k,circ,circx,circy,nt)
c-----------------------------------------------------------------
        call potentials(nn, r(i15), r(i3), r(i4), r(i5), r(i16),
     +  r(i17), r(i18), r(i6), r(i1), r(i2),attack,vinf,
     +  r(i19),k,r(i21),r(i22),r(i23),nt)

c-----------------------------------------------------------------
c       subroutine tanvel(nn,vtan,vinf, theta, attack, r, q, phi,phiold,
c     +       deltat,cp,beta,k,circ,circx,circy,nt,x,y,xbar,ybar,eta)
c-----------------------------------------------------------------
        call tanvel(nn,r(i12),vinf, r(i6), attack,r(i7), r(i15), r(i16),
     +       r(i19),deltat,r(i13),r(i14),k,r(i21),r(i22),r(i23),
     +       nt,r(i1),r(i2),r(i3),r(i4),eta)
c-----------------------------------------------------------------
c       call liftdrag (nn,cp,cl,cd,rlen,vinf,attack,theta,x,y,cm)
c-----------------------------------------------------------------
        call liftdrag (nn,r(i13),cl,cd,r(i5),vinf,attack,r(i6),
     +       r(i1),r(i2),cm,isteady,tsteady,t,ksteady,k,r(i12),
     +       clsteady)
c-----------------------------------------------------------------
c       this ends the potential flow part of the program.  Now, we
c       need to locate all of the important places, istag, ile and ifluid
c-----------------------------------------------------------------
c       call findstag(nn, x, y, vtanp, rlen, istag, B)
c-----------------------------------------------------------------
        call findstag(nn, r(i1), r(i2), r(i12), r(i5), istag, B)
c-----------------------------------------------------------------
c       this will translate the tangential velocities to the boundary
c       layer vtan
c
c       call transfervtan(vtanp, vtanbl, k, istag, nn, nt)
c-----------------------------------------------------------------
        call transfervtan(r(i12), r(i28), k, istag, nn, nt )
        if (isteady .eq. 1) then
c-----------------------------------------------------------------
c       from the input, locate the point at which the deicing fluid
begins.
c       ifluid contains the point in the airfoil geometry at which the
c       deicing fluid should start.  this subroutine also puts the initial
c       fluid depth in the array
c
c       call findh(hxbyc,x,y,nn,ifluid,h,ho,nt)
c-----------------------------------------------------------------
        if (k-ksteady .eq. 0) then
             call findh(hxbyc,r(i3),r(i2),nn,ifluid,r(i42),ho,nt,nl)
        endif
        call setup(nn,nj,istag,r(i5),r(i12),r(i24),r(i28),k,nt,r(i25),
```

```
     +       ratio,r(i42),yftop,yfbottom,ifluid,r(i57),r(i58),
     +       r(i59),r(i60),deltat,ksteady)
c-----------------------------------------------------------------
        if (ivisc .eq. 1) then
c       if (k-ksteady .eq. 2 .and. sor .eq. 0.6) then
c            sor = 0.99
c       endif
        call solvit(k,nn,nj,nt,r(i26),r(i27),r(i28),r(i30),r(i32),
     +       r(i29),r(i31),r(i25),
     +       deltat,r(i24),r(i33), r(i34), r(i35), r(i36), r(i37),
     +       yftop, yfbottom, iyfflag,r(i38),r(i39),r(i40),r(i41),
     +       r(i42),ksteady,ratio,r(i69),r(i70),r(i71),0,
     +       r(i72),r(i73),r(i74),
     +       r(i75),normalflag,isteady,istag,   !this part is for usolve
     +       r(i67),r(i68),                     !these two are misc.
     +       r(i43),r(i44),r(i45),r(i46),       !these are for dfluid
     +       r(i47),r(i48),r(i49),ifluid,nn+1,muratio,r(i76),vinf,t,
     +       markthrough,ineg,icrap,r(i77),r(i78),r(i57),r(i58),
     +       r(i59),r(i60),iseplamt,iseplamb,itransflagt,itransflagbegt,
     +       itransflagb,itransflagbegb,r(i3),sor,inorm,r(i79),vrotate,
     +       attack,ho,r(i80),ifinal,isept,isepb,itsover)
        if (itsover .eq. 1) then
        write(8,755)k,t,vinf,attack,clsteady,
     +       cl,cd,cm,(cl*(0.5)*rho(t)*vinf**2.),
     +       (cd*(0.5)*rho(t)*vinf**2.),r(i21-1+k),
     +       vinf*chord(t)*rho(t)/visc(t),itransflagt,itransflagb,
     +       sor
        print *, 'Program terminated normally.'
        stop
        endif
        itertime = itertime + 1

        if (iseplamb .ne. 0)then
             iseplamt = 0
             iseplamb = 0
             itertime = 0
             normalflag = 0
             ineg = 0
             icrap = 0
             t = t - deltat
c            call normswitch(r(i80),r(i72),nn)
             deltat = deltat*1.03
             print *, 'A change was made in the relaxation factor.'
             print *, 'Extending the time interval.'
             go to 170
        endif

        if (itertime .ge. itertimemax .and. k-ksteady .ne. 0) then
             print *, 'Trying a new solution for vnormal.'
             if (isept .ne. 0 .or. isepb .ne. 0) then
             call normswitch(r(i80),r(i72),nn)
             isept = 0
             isepb = 0
               iseplamt = 0
               iseplamb = 0
```

```
                itertime = 0
                normalflag = 0
                ineg = 0
                icrap = 0
                t = t - deltat
                call normswitch(r(i80),r(i72),nn)
                deltat = deltat*0.98
                print *, 'Separation was predicted.'
              print *, 'Trying to backtrack.'
                go to 170
              endif
              call start3 (nn,vinf,r(i72),r(i3),istag, r(i73),r(i77),
     +             r(i78),sor)
              itertime = 0
              normalflag = 0
              ineg = 0
              icrap = 0
              t = t - deltat
              deltat = deltat*1.01
              sor = sor*0.98
c             call normswitch(r(i80),r(i72),nn)
              go to 170
            endif
            if (normalflag .eq. 1) then
              normalflag = 0
              if (k-ksteady .eq. 0 .and. inorm .eq. 1) then
              go to 444
              endif
              call start2(nn,vinf,r(i72),r(i3),r(i73),attack,sor)
              go to 160
            endif
            endif
            endif
c-----------------------------------------------------------------

 444        itransflagbegt=itransflagt
            itransflagbegb=itransflagb
            call normok(r(i80),r(i72),nn)
            call potoutput(nn,nu,nl,attack,vinf,r(i1),r(i2),r(i3),r(i4),
     +           r(i13),cl,cd,cm,
     +           nacanumber,r(i16),r(i19),r(i15),t,k,r(i21),nt,IGF,
     +           r(i12))
c-----------------------------------------------------------------
c           write the lift and drag coefficients to file
c-----------------------------------------------------------------
            if (k-ksteady .eq. 0. .and. ivisc .eq. 1) then
            call normwriter(nn,r(i72))
            endif
            deltattest = max(2.00*(1./vinf),0.10,abs(r(i21-1+k))/6.00)
c           if (k-ksteady .eq. 1 .and. k .ne. 1) stop
c-----------------------------------------------------------------
c           Question: Will the next time step push the solution beyond
c           13 degrees angle of attack?
c           If so, just extend the last solution...
c-----------------------------------------------------------------
```

```
      call angleofattack(attacktest,alpha0,alphadot,
     +  t+deltattest,trotate)
      if (attacktest .ge. 11.) then
           if (kgood .ne. 0) go to 31
           kgood = k
           temp = 0.01
34         call angleofattack(attacktest,alpha0,alphadot,t+temp,
     +          trotate)
           if (attacktest .ge. 11.) then
                print *, 'Predicting a deltat of :', temp
                deltat = temp
                go to 33
           endif
           temp = temp + 0.01
           go to 34
33         write(8,755)k,t,vinf,attack,clsteady,
     +     cl,cd,cm,(cl*(0.5)*rho(t)*vinf**2.),
     +     (cd*(0.5)*rho(t)*vinf**2.),r(i21-1+k),
     +     vinf*chord(t)*rho(t)/visc(t),itransflagt,itransflagb,
     +     sor
             call normok(r(i80),r(i72),nn)
           call normswitch(r(i80),r(i72),nn)
           print *, 'The next time step will push the angle of attack'
      print *, 'Beyond 11 degrees, so we are excluding the circulation.'
           r(i21-1+k) = 0.
           go to 32
      endif
31    if (kgood .eq. 0) then
      deltat = max(2.00*(1./vinf),0.10,abs(r(i21-1+k))/6.00)
      endif
      if (kgood .ne. 0) then
      deltat = max(2.00*(1./vinf),0.10,abs(r(i21-1+kgood))/25.00)
      print *, 'Using a modified timestep.'
      endif
      if (isteady .eq. 1 .and. ivisc .eq. 1) then
      call hwriter(nn,r(i3),t,r(i42),ifluid)
      call tautopwriter(nn,r(i3),t,r(i33),istag)
      call deltatopwriter(nn,r(i3),t,r(i34),istag)
      endif
      write(8,755)k,t,vinf,attack,clsteady,
     +  cl,cd,cm,(cl*(0.5)*rho(t)*vinf**2.),
     +       (cd*(0.5)*rho(t)*vinf**2.),r(i21-1+k),
     +       vinf*chord(t)*rho(t)/visc(t),itransflagt,itransflagb,
     +       sor
 32   call reload(nn,nj,r(i26),r(i27),r(i28),r(i33),r(i34),r(i35),
     +  r(i36),
     +  r(i37),r(i42),r(i76))
 100  continue
      close(unit=8)
      close(unit=12)
      close(unit=72)
      close(unit=73)
      close(unit=74)
c     close(unit=75)
c     close(unit=76)
```

```
c       close(unit=77)
c       close(unit=83)
c       close(unit=84)
c       close(unit=85)
c       close(unit=86)
c       close(unit=87)

 701    print *,'Program terminated normally.'

 101    format(' Do you need to create the airfoil geometry ',
     +  '(1) Yes (2) No ?',$)
 102    format(' Do you need to create the input parameters for the ',
     +  'boundary ',
     +  'layer (1) Yes (2) No ?',$)
 103    format(' Do you need to input the control parameters ',
     +  '(1) Yes (2) No ?',$)


 600    format(1x, i5)
 700    format(1x, f10.4)
 755    format(1x,i5,10(5x,f15.7),5x,f12.2,2(5x,i5),5x,f12.2)
 750    format(1x,3x,'Time Step',5x,'Time',12x,'Vinf',12x,'alpha',12x,
     +       'clsteady',12x,'cl',12x,'cd',12x,'cm',
     +  12x,'l',12x,'d',
     +       12x,'circ',
     +       9x,'Reynolds Number',8x,'Trans_node_bot',8x,'Trans_node_top',
     +       8x, 'Relaxation factor')

       end
c-----------------------------------------------------------------
c       subroutine coef
c
c       sets up the a matrix for the geometry
c-----------------------------------------------------------------
       subroutine coef(xbar,ybar,x,y,rlen,theta,r,beta,a,nn,eta)

       dimension xbar(nn),ybar(nn),x(nn+1),y(nn+1),theta(nn)
       dimension r(nn,nn+1), beta(nn,nn), a(nn+1,nn+1),rlen(nn)
       pi = atan(1.)*4.
      kutta = nn +1
c-----------------------------------------------------------------
c       Calculate the midpoints of the panels
c-----------------------------------------------------------------
       do  10 i = 1, nn
            xbar (i) = (x(i) + x(i+1))/ 2.
            ybar (i) = (y(i) + y(i+1))/ 2.
 10    continue
c-----------------------------------------------------------------
c       Find the length of each panel and the orientation
c-----------------------------------------------------------------
       do 20 i = 1, nn
            dy = (y (i+1) - y (i))
            dx = (x (i+1) - x (i))
            rlen(i) = sqrt ((dy * dy) + (dx * dx))
            theta(i) = atan2(dy,dx)
```

```
 20      continue
c-----------------------------------------------------------------
c        Find the distances from the midpoints to the nodes
c-----------------------------------------------------------------
         do 40 i = 1, nn
              do 50 j = 1, nn
                   dy = y(j) - ybar(i)
                   dx = x(j) - xbar(i)
                   r(i,j) = sqrt((dy*dy) + (dx*dx))
                   dy = y(j+1) - ybar(i)
                   dx = x(j+1) - xbar(i)
                   r(i,j+1) = sqrt((dy*dy) + (dx*dx))
 50           continue
 40      continue
c-----------------------------------------------------------------
c        Find the inclusive angles
c-----------------------------------------------------------------
         do 60 i = 1, nn
              do 70 j = 1, nn
              if (i .eq. j) then
                   beta(i,i) = pi
              else
                   rone = ybar(i) - y(j+1)
                   two = xbar(i) - x(j)
                   three = xbar(i) - x(j+1)
                   four = ybar(i) - y(j)
                   rnum = (rone * two) - (three * four)
                   den = (three * two) + (rone * four)
                   beta(i,j) = atan2(rnum,den)
              endif
 70           continue
 60      continue

c-----------------------------------------------------------------
c        Create the a matrix, first the a(i,j) terms
c-----------------------------------------------------------------
         do 80 i = 1, nn
         do 90 j = 1, nn
              rone = (sin(theta(i) -theta(j)) / (2 *pi))
              two = (cos(theta(i) -theta(j)) / (2 *pi))
              three = alog (r(i,j+1)/r(i,j))
              a(i,j) = (rone * three) + (two * beta(i,j))
 90      continue
 80      continue
c-----------------------------------------------------------------
c        Now, the a(i,kutta) terms
c-----------------------------------------------------------------
         do 100 i = 1, nn
              a(i, kutta) = 0
              do 110 j =1, nn
                   rone = alog(r(i,j+1)/r(i,j))
                   two = cos(theta(i) - theta(j))
                   three = sin(theta(i) - theta(j))
                   four = beta(i,j)* three
                   five = rone * two
```

```
                a(i, kutta) = a(i, kutta) + ((1/(2* pi)) * (five -
     +                      four))
 110         continue
 100     continue
c-----------------------------------------------------------------
c        Now, the a(kutta,j) terms
c-----------------------------------------------------------------
         do 120 j = 1, nn
              a(kutta, j) = 0
              do 130 k = 1, nn, nn - 1
                   rone = cos(theta(k) -theta(j)) * alog(r(k,j+1)/
     +                  r(k,j))
                   two = beta(k,j) * sin(theta(k) -theta(j))
                   a(kutta, j) = a(kutta,j) + ((1/(2*pi))*(two - rone))
 130          continue
 120     continue
c-----------------------------------------------------------------
c        Now, the a(kutta,kutta) term
c-----------------------------------------------------------------
         a(kutta, kutta) =0.0
         do 140 k = 1, nn, nn -  1
              do 150 j = 1, nn
                   rone = cos(theta(k) -theta(j)) * beta(k,j)
                   two = sin (theta(k) -theta(j)) * alog(r(k,j+1)/
     +                  r(k,j))
                   three = (1/(2*pi)) * (rone + two)
                   a(kutta, kutta) = a(kutta, kutta) + three
 150          continue
 140     continue
         eta = (theta(1) + theta(nn) + pi)/2.
         return
         end
c-----------------------------------------------------------------
c        subroutine control
c
c        this subroutine gets the control parameters that
c        govern the potential flow and the deicing fluid
c-----------------------------------------------------------------
         subroutine control(ho, hxbyc, alpha0, alphadot, vinf0,
     +   vrotate, vinfdot, muratio,ivisc,inorm)
         real muratio


 130     print *, 'Please input the ratio of the initial hieght of deicing'
         print *, 'fluid to the length of the airfoil.'
         read *, ho
         if (ho .gt. 1.0) then
              print *, 'Ratio must be less than one.'
              go to 130
         endif
         if (ho .lt. 0.0) then
              print *, 'Give it some initial height.'
              go to 130
         endif
```

```
140     print *, 'In terms of the nondimensional chord length',','
        print *, 'on the top of the airfoil',',','at what percent'
        print *, 'of chord do you wish the deicing fluid to begin?'
        read *, hxbyc
        if (hxbyc .gt. 1.0) then
             print *, 'Ratio must be less than one.'
             go to 130
        endif
        if (hxbyc .lt. 0.0) then
             print *, 'Ratio must be greater than zero.'
             go to 130
        endif

75      print *, 'Please input the initial angle of attack',
    +    ' in degrees.'
        read *, alpha0
        if (abs(alpha0) .ge. 45.) then
        go to 75
        endif

175     print *, 'Please input the constant pitch rate for the',
    +    ' angle of attack in degrees per second.'
        read *, alphadot
        if (abs(alphadot) .ge. 45.) then
        go to 175
        endif
275     print *, 'Please input the starting speed V0 in units',
    +    ' length per second.'
        read *, vinf0
        if (vinf0 .le. 1.) then
        go to 275
        endif
475     print *, 'Please input the rotation speed vrotate in units',
    +    ' length per second.'
        read *, vrotate
        if (vrotate .le. vinf0) then
        print *, 'Rotation speed must be larger than'
        print *, 'initial speed.'
        go to 475
        endif

        print *, 'Please input acceleration rate of the freestream',
    +    ' in units length per second per second.'
        read *, vinfdot

375     print *, 'Please input the ratio of viscosities for the fluid',
    +    ' muf/muair.'
        read *, muratio
        if (vinf0 .le. 1.) then
        go to 375
        endif
575     print *, 'Would you like this to be a (1) viscous anlaysis or',
    +    ' (2) inviscid analysis?'
        read *, ivisc
        if (ivisc .ne. 1 .and. ivisc .ne. 2) then
```

```
        go to 575
        endif
675     print *, 'Would you like to read the initial normal velocities',
     +  ' from a file? (1) Yes or (2) No'
        read *, inorm
        if (inorm .ne. 1 .and. inorm .ne. 2) then
        go to 675
        endif


        open (unit = 7, file = 'contr.dat', status = 'unknown')
        write(7,700) ho
        write(7,700) hxbyc
        write(7,700) alpha0
        write(7,700) alphadot
        write(7,700) vinf0
        write(7,700) vrotate
        write(7,700) vinfdot
        write(7,700) muratio
        write(7,800) ivisc
        write(7,800) inorm
        close(unit = 7)

 700    format(1x, f10.4)
 800    format(1x, i5)
        return
        end
c-----------------------------------------------------------------
        subroutine getcontrol(ho, hxbyc, alpha0, alphadot, vinf0,
     +  vrotate, vinfdot, muratio,ivisc,inorm)
        real muratio
        open (unit = 7, file = 'contr.dat', status = 'unknown')
        read (7,700) ho
        read (7,700) hxbyc
        read (7,700) alpha0
        read (7,700) alphadot
        read (7,700) vinf0
        read (7,700) vrotate
        read (7,700) vinfdot
        read (7,700) muratio
        read (7,800) ivisc
        read (7,800) inorm
        close(unit = 7)

 700    format(1x, f10.4)
 800    format(1x, i5)
        return
        end

        subroutine cpvdatapoints(nu,nl,nn,xbar,ybar,iASDIS,
     +  iSOLAT75,iSOLAB75,
     +  iSOLAT50,iSOLAB50,iSOLAT13,iSOLAB13,iSOLAT08,iSOLAB08)
        dimension xbar(nn), ybar(nn)
c-----------------------------------------------------------------
c       find the ASDIS port
```

```
c-----------------------------------------------------------------
      iASDIS = nl
c-----------------------------------------------------------------
c     find the SOLA ports at 75, 50, 13, 8 % of chord
c-----------------------------------------------------------------


      do 10 i=1,nl
      if (xbar(i) .le. 0.75) then
      diff = abs(xbar(i) -0.75)
      if (abs(xbar(i-1) -0.75) .le. diff) then
      iSOLAB75 = i-1
      go to 20
      endif
      iSOLAB75 = i
      go to 20
      endif
10    continue


20    do 30 i=1,nl
      if (xbar(i) .le. 0.50) then
      diff = abs(xbar(i) -0.50)
      if (abs(xbar(i-1) -0.50) .le. diff) then
      iSOLAB50 = i-1
      go to 40
      endif
      iSOLAB50 = i
      go to 40
      endif
30    continue
40    do 50 i=1,nl
      if (xbar(i) .le. 0.13) then
      diff = abs(xbar(i) -0.13)
      if (abs(xbar(i-1) -0.13) .le. diff) then
      iSOLAB13 = i-1
      go to 60
      endif
      iSOLAB13 = i
      go to 60
      endif
50    continue
60    do 70 i=1,nl
      if (xbar(i) .le. 0.08) then
      diff = abs(xbar(i) -0.08)
      if (abs(xbar(i-1) -0.08) .le. diff) then
      iSOLAB08 = i-1
      go to 80
      endif
      iSOLAB08 = i
      go to 80
      endif
70    continue
```

```
80      do 90 i=nl+1,nn
        if (xbar(i) .ge. 0.75) then
        diff = abs(xbar(i) -0.75)
        if (abs(xbar(i+1) -0.75) .le. diff) then
        iSOLAT75 = i+1
        go to 100
        endif
        iSOLAT75 = i
        go to 100
        endif
90      continue
100     do 110 i=nl+1,nn
        if (xbar(i) .ge. 0.50) then
        diff = abs(xbar(i) -0.50)
        if (abs(xbar(i+1) -0.50) .le. diff) then
        iSOLAT50 = i+1
        go to 120
        endif
        iSOLAT50 = i
        go to 120
        endif
110     continue
120     do 130 i=nl+1,nn
        if (xbar(i) .ge. 0.13) then
        diff = abs(xbar(i) -0.13)
        if (abs(xbar(i+1) -0.13) .le. diff) then
        iSOLAT13 = i+1
        go to 140
        endif
        iSOLAT13 = i
        go to 140
        endif
130     continue
140     do 150 i=nl+1,nn
        if (xbar(i) .ge. 0.08) then
        diff = abs(xbar(i) -0.08)
        if (abs(xbar(i+1) -0.08) .le. diff) then
        iSOLAT08 = i+1
        go to 160
        endif
        iSOLAT08 = i
        go to 160
        endif
150     continue
160     open(unit = 12, file='cpv.dat', status='unknown')
        write (12,750) iASDIS, xbar(iASDIS), ybar(iASDIS)
        write (12,751) iSOLAT08, xbar(iSOLAT08), ybar(iSOLAT08)
        write (12,752) iSOLAB08, xbar(iSOLAB08), ybar(iSOLAB08)
        write (12,753) iSOLAT13, xbar(iSOLAT13), ybar(iSOLAT13)
        write (12,754) iSOLAB13, xbar(iSOLAB13), ybar(iSOLAB13)
        write (12,755) iSOLAT50, xbar(iSOLAT50), ybar(iSOLAT50)
        write (12,756) iSOLAB50, xbar(iSOLAB50), ybar(iSOLAB50)
        write (12,757) iSOLAT75, xbar(iSOLAT75), ybar(iSOLAT75)
        write (12,758) iSOLAB75, xbar(iSOLAB75), ybar(iSOLAB75)
        write (12,*) ' '
```

```
      write (12,760)

750   format(1x,'ASDIS port location:',i5,1x,'at xbar = ',f12.4,1x,
     +  'and ybar = ', f12.4)
751   format(1x,'Top 8% port location:',i5,1x,'at xbar = ',f12.4,1x,
     +  'and ybar = ', f12.4)
752   format(1x,'Bottom 8% port location:',i5,1x,'at xbar = ',f12.4,1x,
     +  'and ybar = ', f12.4)
753   format(1x,'Top 13% port location:',i5,1x,'at xbar = ',f12.4,1x,
     +  'and ybar = ', f12.4)
754   format(1x,'Bottom 13% port location:',i5,1x,'at xbar = ',f12.4,1x,
     +  'and ybar = ', f12.4)
755   format(1x,'Top 50% port location:',i5,1x,'at xbar = ',f12.4,1x,
     +  'and ybar = ', f12.4)
756   format(1x,'Bottom 50% port location:',i5,1x,'at xbar = ',f12.4,1x,
     +  'and ybar = ', f12.4)
757   format(1x,'Top 75% port location:',i5,1x,'at xbar = ',f12.4,1x,
     +  'and ybar = ', f12.4)
758   format(1x,'Bottom 75% port location:',i5,1x,'at xbar = ',f12.4,1x,
     +  'and ybar = ', f12.4)

760   format(1x'  Time',3x,'CpASDSIS',3x,'Cp8t',3x,'Cp8b',3x,
     +  'Cp13t',3x,'Cp13b',
     +  3x,'Cp50t',3x,'Cp50b',3x,'Cp75t',3x,'Cp75b',3x,'dCp8',3x,'dCp13',
     +  3x,'dCp50',3x,'dCp75',3x,'Cpv8',3x,'Cpv13',3x,'Cpv50',3x,'Cpv75',
     +  3x,'Vinf')
500   return
      end

      subroutine cpvoutput(nu,nl,nn,iASDIS,iSOLAT75,iSOLAB75,
     +  iSOLAT50,iSOLAB50,iSOLAT13,iSOLAB13,iSOLAT08,iSOLAB08,cp,
     +  t,vinf)
      dimension cp(nn)
      write (12,600) t,cp(iASDIS),cp(iSOLAT08), cp(iSOLAB08),
     +      cp(iSOLAT13), cp(iSOLAB13),cp(iSOLAT50), cp(iSOLAB50),
     +      cp(iSOLAT75), cp(iSOLAB75),cp(iSOLAB08)-cp(iSOLAT08),
     +      cp(iSOLAB13)-cp(iSOLAT13),cp(iSOLAB50)-cp(iSOLAT50),
     +      cp(iSOLAB75)-cp(iSOLAT75),
     +      cp(iSOLAB08)-cp(iSOLAT08)/cp(iASDIS),
     +      cp(iSOLAB13)-cp(iSOLAT13)/cp(iASDIS),
     +      cp(iSOLAB50)-cp(iSOLAT50)/cp(iASDIS),
     +      cp(iSOLAB75)-cp(iSOLAT75)/cp(iASDIS), vinf
600   format (1x,19(3x,f12.7))
      return
      end
c-----------------------------------------------------------------
c     writing the output data to file
c
c     subroutine hdatapoints
c-----------------------------------------------------------------
      subroutine hdatapoints(nn,xbar,ih25,ih50,ih75,ih100,ifluid,h,x)
      dimension h(nn+1,2),xbar(nn+1)
      open (unit = 27, file = 'houtput.dat', status = 'unknown')
      write(27,400) ifluid
```

```
      do 10 m = 1, 4
      do 20 i = ifluid, n+1
      if ((xbar(i)) .ge. (0.249 * float(m)) ) then
      if (m .eq. 1) then
           i1 = i
           go to 10
           elseif (m .eq. 2) then
           i2 = i
           go to 10
           elseif (m .eq. 3) then
           i3 = i
           go to 10
           elseif (m .eq. 4) then
           i4 = i
      endif
      endif
20    continue
10    continue
      write(7, 701) xbar(i1)
      write(7, 702) xbar(i2)
      write(7, 703) xbar(i3)
      write(7, 704) xbar(i4)
      write(7,*) ' '
      write(7,700)
      do 510 m = 1, k
      write(7,600) float(m-1)*deltat,h(i1,m),h(i2,m),h(i3,m),h(i4,m)
510   continue

400   format(1x,'Fluid begins at node ',i5)

600   format(1x,5(3x,f15.7))
700   format(1x,5x,'Time',5x,'h at 0.25',5x,'h at 0.50',
     +  5x,'h at 0.75',5x, 'h at 1.00')
701   format ('Information at x1 =',2x,f12.8)
702   format ('Information at x2 =',2x,f12.8)
703   format ('Information at x3 =',2x,f12.8)
704   format ('Information at x4 =',2x,f12.8)
      return
      end
c-----------------------------------------------------------------
c     writing the output data to file
c
c     subroutine houtput
c-----------------------------------------------------------------
      subroutine houtput(k,ni,nt,h,x,deltat,nj,istart,istop,isign)
      dimension h(ni,2),x(ni,nj)
      open (unit = 27, file = 'houtput.dat', status = 'unknown')
      write(7,400) k
      write(7,450) deltat
      write(7,451) deltat*k
      do 10 m = 1, 4
      do 20 i = istart+isign, istop, isign
      if (abs(x(i,1)) .ge. (0.249 * float(m)) ) then
      if (m .eq. 1) then
           i1 = i
```

```
            go to 10
            elseif (m .eq. 2) then
            i2 = i
            go to 10
            elseif (m .eq. 3) then
            i3 = i
            go to 10
            elseif (m .eq. 4) then
            i4 = i
        endif
        endif
20      continue
10      continue
        write(7, 701) x(i1,1)
        write(7, 702) x(i2,1)
        write(7, 703) x(i3,1)
        write(7, 704) x(i4,1)
        write(7,*) ' '
        write(7,700)
        do 510 m = 1, k
        write(7,600) float(m-1)*deltat,h(i1,m),h(i2,m),h(i3,m),h(i4,m)
510     continue
        close (unit = 7)
400     format(1x,'Values of h for time step',i5)
450     format(1x,'Time step:',f12.4)
451     format(1x,'Time:',f12.4)
600     format(1x,5(3x,f15.7))
700     format(1x,5x,'Time',5x,'h at 0.25',5x,'h at 0.50',
    +   5x,'h at 0.75',5x, 'h at 1.00')
701     format ('Information at x1 =',2x,f12.8)
702     format ('Information at x2 =',2x,f12.8)
703     format ('Information at x3 =',2x,f12.8)
704     format ('Information at x4 =',2x,f12.8)
        return
        end
c-----------------------------------------------------------------
c       subroutine createinput
c
c       This subroutine inputs the required parameters.
c-----------------------------------------------------------------
        subroutine createinput(nj,yf,deltat,tfinal,ratio)
c-----------------------------------------------------------------
c       getting number of nodes on top and bottom surfaces
c-----------------------------------------------------------------

 90     print *, 'Please input the number of nodes in the y direction.'
        print *, 'This is in the direction perpendicular to the '
        print *, 'boundary layer.'
        read *, nj
        if (nj .gt. 300) then
                print *, 'Too many nodes...'
                go to 90
        elseif (nj .le. 3) then
                print *, 'Need more nodes...'
                go to 90
```

```
      endif

110   print *, 'Please input the non-dimensional height of the region'
      print *, 'of interest in the y direction.'
      read *, yf
      if (yf .le. 0.00 .or. yf .ge. 5.0) then
      go to 110
      endif

100   print *, 'Please input the time interval between calculations.'
      read *, deltat
      print *, 'Please input the final time for calculations to finish.'
      read *, tfinal
      if (deltat .le. 0.0) go to 100
120   print *, 'Please input the ratio of the next dy to the current dy'
      print *, 'in the boundary layer.'
      read *, ratio
      if (ratio .lt. 1.0) then
           print *, 'Ratio must be greater than or equal to one.'
           go to 120
      endif

      open (unit = 7, file = 'input.dat', status = 'unknown')
      write(7,600) nj
      write(7,700) yf
      write(7,700) deltat
      write(7,700) tfinal
      write(7,700) ratio
      close(unit = 7)
600   format(1x, i5)
700   format(1x, f10.4)
      return
      end
c-----------------------------------------------------------------
c     subroutine getinputparameters(nj,yf,deltat,tfinal,ratio)
c-----------------------------------------------------------------
      subroutine getinputparameters(nj,yf,deltat,tfinal,ratio)

      open (unit = 7, file = 'input.dat', status = 'unknown')
      read(7,600) nj
      read(7,700) yf
      read(7,700) deltat
      read(7,700) tfinal
      read(7,700) ratio
      close(unit = 7)
600   format(1x, i5)
700   format(1x, f10.4)
      return
      end
      SUBROUTINE SVDFIT(X,Y,SIG,NDATA,A,MA,U,V,W,MP,NP,CHISQ,FUNCS)
      DIMENSION X(NDATA),Y(NDATA),SIG(NDATA),A(MA),V(NP,NP),
     *  U(MP,NP),W(NP),B(1000),AFUNC(50)
      EXTERNAL funcs
      DO 12 I=1,NDATA
      CALL FUNCS(X(I),AFUNC,MA)
```

```
        TMP=1./SIG(I)
        DO 11 J=1,MA
             U(I,J)=AFUNC(J)*TMP
11      CONTINUE
        B(I)=Y(I)*TMP
12      CONTINUE
        CALL SVDCMP(U,NDATA,MA,MP,NP,W,V)
        WMAX=0.
        DO 13 J=1,MA
             IF(W(J).GT.WMAX)WMAX=W(J)
13      CONTINUE
      THRESH=1.E-10*WMAX
      DO 14 J=1,MA
        IF(W(J).LT.THRESH)W(J)=0.
14    CONTINUE
      CALL SVBKSB(U,W,V,NDATA,MA,MP,NP,B,A)
      CHISQ=0.
      DO 16 I=1,NDATA
        CALL FUNCS(X(I),AFUNC,MA)
        SUM=0.
        DO 15 J=1,MA
          SUM=SUM+A(J)*AFUNC(J)
15      CONTINUE
        CHISQ=CHISQ+((Y(I)-SUM)/SIG(I))**2
16    CONTINUE
      RETURN
      END

        SUBROUTINE SVDCMP(A,M,N,MP,NP,W,V)
      DIMENSION A(MP,NP),W(NP),V(NP,NP),RV1(100)
      G=0.0
      SCALE=0.0
      ANORM=0.0
      DO 25 I=1,N
        L=I+1
        RV1(I)=SCALE*G
        G=0.0
        S=0.0
        SCALE=0.0
        IF (I.LE.M) THEN
          DO 11 K=I,M
            SCALE=SCALE+ABS(A(K,I))
11        CONTINUE
          IF (SCALE.NE.0.0) THEN
            DO 12 K=I,M
              A(K,I)=A(K,I)/SCALE
              S=S+A(K,I)*A(K,I)
12          CONTINUE
            F=A(I,I)
            G=-SIGN(SQRT(S),F)
            H=F*G-S
            A(I,I)=F-G
            IF (I.NE.N) THEN
              DO 15 J=L,N
                S=0.0
```

```
                DO 13 K=I,M
                  S=S+A(K,I)*A(K,J)
13              CONTINUE
                F=S/H
                DO 14 K=I,M
                  A(K,J)=A(K,J)+F*A(K,I)
14              CONTINUE
15            CONTINUE
            ENDIF
            DO 16 K= I,M
              A(K,I)=SCALE*A(K,I)
16          CONTINUE
          ENDIF
        ENDIF
        W(I)=SCALE *G
        G=0.0
        S=0.0
        SCALE=0.0
        IF ((I.LE.M).AND.(I.NE.N)) THEN
          DO 17 K=L,N
            SCALE=SCALE+ABS(A(I,K))
17        CONTINUE
          IF (SCALE.NE.0.0) THEN
            DO 18 K=L,N
              A(I,K)=A(I,K)/SCALE
              S=S+A(I,K)*A(I,K)
18          CONTINUE
            F=A(I,L)
            G=-SIGN(SQRT(S),F)
            H=F*G-S
            A(I,L)=F-G
            DO 19 K=L,N
              RV1(K)=A(I,K)/H
19          CONTINUE
            IF (I.NE.M) THEN
              DO 23 J=L,M
                S=0.0
                DO 21 K=L,N
                  S=S+A(J,K)*A(I,K)
21              CONTINUE
                DO 22 K=L,N
                  A(J,K)=A(J,K)+S*RV1(K)
22              CONTINUE
23            CONTINUE
            ENDIF
            DO 24 K=L,N
              A(I,K)=SCALE*A(I,K)
24          CONTINUE
          ENDIF
        ENDIF
        ANORM=MAX(ANORM,(ABS(W(I))+ABS(RV1(I))))
25    CONTINUE
      DO 32 I=N,1,-1
        IF (I.LT.N) THEN
          IF (G.NE.0.0) THEN
```

```
            DO 26 J=L,N
              V(J,I)=(A(I,J)/A(I,L))/G
26          CONTINUE
            DO 29 J=L,N
              S=0.0
              DO 27 K=L,N
                S=S+A(I,K)*V(K,J)
27            CONTINUE
              DO 28 K=L,N
                V(K,J)=V(K,J)+S*V(K,I)
28            CONTINUE
29          CONTINUE
          ENDIF
          DO 31 J=L,N
            V(I,J)=0.0
            V(J,I)=0.0
31        CONTINUE
        ENDIF
        V(I,I)=1.0
        G=RV1(I)
        L=I
32    CONTINUE
      DO 39 I=N,1,-1
        L=I+1
        G=W(I)
        IF (I.LT.N) THEN
          DO 33 J=L,N
            A(I,J)=0.0
33        CONTINUE
        ENDIF
        IF (G.NE.0.0) THEN
          G=1.0/G
          IF (I.NE.N) THEN
            DO 36 J=L,N
              S=0.0
              DO 34 K=L,M
                S=S+A(K,I)*A(K,J)
34            CONTINUE
              F=(S/A(I,I))*G
              DO 35 K=I,M
                A(K,J)=A(K,J)+F*A(K,I)
35            CONTINUE
36          CONTINUE
          ENDIF
          DO 37 J=I,M
            A(J,I)=A(J,I)*G
37        CONTINUE
        ELSE
          DO 38 J= I,M
            A(J,I)=0.0
38        CONTINUE
        ENDIF
        A(I,I)=A(I,I)+1.0
39    CONTINUE
      DO 49 K=N,1,-1
```

```
      DO 48 ITS=1,30
        DO 41 L=K,1,-1
          NM=L-1
          IF ((ABS(RV1(L))+ANORM).EQ.ANORM)  GO TO 2
          IF ((ABS(W(NM))+ANORM).EQ.ANORM)  GO TO 1
41      CONTINUE
1       C=0.0
        S=1.0
        DO 43 I=L,K
          F=S*RV1(I)
          IF ((ABS(F)+ANORM).NE.ANORM) THEN
            G=W(I)
            H=SQRT(F*F+G*G)
            W(I)=H
            H=1.0/H
            C= (G*H)
            S=-(F*H)
            DO 42 J=1,M
              Y=A(J,NM)
              Z=A(J,I)
              A(J,NM)=(Y*C)+(Z*S)
              A(J,I)=-(Y*S)+(Z*C)
42          CONTINUE
          ENDIF
43      CONTINUE
2       Z=W(K)
        IF (L.EQ.K) THEN
          IF (Z.LT.0.0) THEN
            W(K)=-Z
            DO 44 J=1,N
              V(J,K)=-V(J,K)
44          CONTINUE
          ENDIF
          GO TO 3
        ENDIF
        IF (ITS.EQ.30) then !PAUSE 'No convergence in 30 iterations'
        endif
        X=W(L)
        NM=K-1
        Y=W(NM)
        G=RV1(NM)
        H=RV1(K)
        F=((Y-Z)*(Y+Z)+(G-H)*(G+H))/(2.0*H*Y)
        G=SQRT(F*F+1.0)
        F=((X-Z)*(X+Z)+H*((Y/(F+SIGN(G,F)))-H))/X
        C=1.0
        S=1.0
        DO 47 J=L,NM
          I=J+1
          G=RV1(I)
          Y=W(I)
          H=S*G
          G=C*G
          Z=SQRT(F*F+H*H)
          RV1(J)=Z
```

```
            C=F/Z
            S=H/Z
            F= (X*C)+(G*S)
            G=-(X*S)+(G*C)
            H=Y*S
            Y=Y*C
            DO 45 NM=1,N
              X=V(NM,J)
              Z=V(NM,I)
              V(NM,J)= (X*C)+(Z*S)
              V(NM,I)=-(X*S)+(Z*C)
45          CONTINUE
            Z=SQRT(F*F+H*H)
            W(J)=Z
            IF (Z.NE.0.0) THEN
              Z=1.0/Z
              C=F*Z
              S=H*Z
            ENDIF
            F= (C*G)+(S*Y)
            X=-(S*G)+(C*Y)
            DO 46 NM=1,M
              Y=A(NM,J)
              Z=A(NM,I)
              A(NM,J)= (Y*C)+(Z*S)
              A(NM,I)=-(Y*S)+(Z*C)
46            CONTINUE
47          CONTINUE
            RV1(L)=0.0
            RV1(K)=F
            W(K)=X
48        CONTINUE
3         CONTINUE
49      CONTINUE
        RETURN
        END
        SUBROUTINE SVBKSB(U,W,V,M,N,MP,NP,B,X)
        DIMENSION U(MP,NP),W(NP),V(NP,NP),B(MP),X(NP),TMP(100)
        DO 12 J=1,N
          S=0.
          IF(W(J).NE.0.)THEN
            DO 11 I=1,M
              S=S+U(I,J)*B(I)
11          CONTINUE
            S=S/W(J)
          ENDIF
          TMP(J)=S
12      CONTINUE
        DO 14 J=1,N
          S=0.
          DO 13 JJ=1,N
            S=S+V(J,JJ)*TMP(JJ)
13        CONTINUE
          X(J)=S
14      CONTINUE
```

```
      RETURN
      END

         subroutine FUNCS(x,AFUNC,ma)
         real x
         dimension AFUNC(3)
         AFUNC(1) = 1.
         AFUNC(2) = x
         AFUNC(3) = x**2.
         return
         end
c-----------------------------------------------------------------
c        subroutine cylinder
c
c        cylinder makes the points for the x and y positions of the
c        cylinder geometry and is called by grid
c
c-----------------------------------------------------------------
         subroutine cylinder(nl,nu,nn,x,y,AR,title,IGF)
         dimension x(nn+1),y(nn+1)
         character *40, title

         pi = atan(1.)*4.

         do 10 i = 1, nl
         fract = float(i-1)/float(nl)
         x(i) = 0.5 * (1. + cos(-fract*pi))
         y(i) = -(0.5/AR) * sin(fract*pi)
 10      continue

         do 20 i = nl+1, nn
         fract = float(i-1-nl)/float(nu)
         x(i) = 0.5 * (1. - cos(fract*pi))
         y(i) = (0.5/AR) * sin(fract*pi)
 20      continue
         x(nn+1) = x(1)
         y(nn+1) = y(1)

         if (AR .eq. 1.0) then
         print *, 'Geometry for cylinder complete.'
         go to 50
         endif

         print *, 'Geometry for ellipse complete.'

 50      open (unit = 7, file = 'xy.dat', status = 'old')
c-----------------------------------------------------------------
c        write to the file 'xy.dat'
c-----------------------------------------------------------------
         write(7,103) title
         write(7,601) nl
         write(7,601) nu
         write(7,601) nn
         write(7,601) IGF
 103     format(A)
```

```
 601     format(1x, i5)
         do 510 i = 1, nn+1
         write(7,600) x(i),y(i)
 510     continue
         print *, 'Data written to file xy.dat.'
         close (unit = 7)
 600     format(1x,2(3x,f12.7))
 700     format(1x, f10.4)
         return
         end
*****************************************************************
         subroutine decomp(ndim,n,a,condt,ipvt,work)
*****************************************************************
         dimension a(ndim,n),work(n),ipvt(n)
         ipvt(n)=1
         if (n.eq.1) go to 80
         nm1=n-1
         anorm=0.0
         do 10 j=1,n
           t=0.0
           do 5 i=1,n
              t=t+abs(a(i,j))
 5         continue
           if (t.gt.anorm) anorm=t
 10      continue
         do 35 k=1,nm1
           kp1=k+1
           m=k
           do 15 i=kp1,n
              if (abs(a(i,k)).gt.abs(a(m,k))) m=i
 15        continue
           ipvt(k)=m
           if (m.ne.k) ipvt(n)=-ipvt(n)
           t=a(m,k)
           a(m,k)=a(k,k)
           a(k,k)=t
           if (t.eq.0.0) go to 35
           do 20 i=kp1,n
              a(i,k)=-a(i,k)/t
 20        continue
           do 30 j=kp1,n
              t=a(m,j)
              a(m,j)=a(k,j)
              a(k,j)=t
              if (t.eq.0.0) go to 30
              do 25 i=kp1,n
               a(i,j)=a(i,j)+a(i,k)*t
 25           continue
 30        continue
 35      continue
         do 50 k=1,n
           t=0.0
           if (k.eq.1) go to 45
           km1=k-1
           do 40 i=1,km1
```

```
            t=t+a(i,k)*work(i)
40        continue
45        ek=1.0
          if (t.lt.0.0) ek=-1.0
          if (a(k,k).eq.0.0) go to 90
          work(k)=-(ek+t)/a(k,k)
50      continue
        do 60 kb=1,nm1
          k=n-kb
          t=0.0
          kp1=k+1
          do 55 i=kp1,n
             t=t+a(i,k)*work(k)
55        continue
          work(k)=t
          m=ipvt(k)
          if (m.eq.k) go to 60
          t=work(m)
          work(m)=work(k)
          work(k)=t
60      continue
        ynorm=0.0
        do 65 i=1,n
          ynorm=ynorm+abs(work(i))
65      continue
        call solve(ndim,n,a,work,ipvt)
        znorm=0.0
        do 70 i=1,n
          znorm=znorm+abs(work(i))
70      continue
        condt=anorm*znorm/ynorm
        if (condt.lt.1.0) condt=1.0
        return
80      condt=1.0
        if (a(1,1).ne.0.0) return
90      condt=1.0e+32
        return
        end

*****************************************************************
        subroutine solve(ndim,n,a,b,ipvt)
*****************************************************************
        dimension a(ndim,n),b(n),ipvt(n)
        if (n.eq.1) go to 50
        nm1=n-1
        do 20 k=1,nm1
          kp1=k+1
          m=ipvt(k)
          t=b(m)
          b(m)=b(k)
          b(k)=t
          do 10 i=kp1,n
             b(i)=b(i)+a(i,k)*t
10        continue
20      continue
```

```
        do 40 kb=1,nm1
          km1=n-kb
          k=km1+1
          b(k)=b(k)/a(k,k)
          t=-b(k)
          do 30 i=1,km1
             b(i)=b(i)+a(i,k)*t
30           continue
40      continue
50      b(1)=b(1)/a(1,1)
        return
        end

        subroutine DERIVS(x,y,dydx,acc)
        real y(10),dydx(10)
        real x, acc
        dydx(1) = y(2)
        dydx(2) = y(3)
        dydx(3) = y(2)*y(2) - 1. - y(1)*y(3) - acc + (acc*y(2))
        return
        end

c-----------------------------------------------------------------------
c       subroutine dfluid(k,nn,nt,h,tau,vtan,f,aa,bb,dd,
c    +  work1,work2, work3,x,deltat,u,nj,ifluid,istop)
c
c       this is a subroutine that will solve the partial differential
c       equation developled by djcronin
c
c       dh/dt + d/dx(((tau (h^2))/2 muf) - (dp/dx) (h^3))/3 muf)) = 0
c
c       Copyright Dennis J. Cronin, 1994
c       Why, because I said so...
c-----------------------------------------------------------------------
        subroutine dfluid(k,nn,nt,h,tau,vtan,f,aa,bb,dd,
     +  work1,work2, work3,x,deltat,u,nj,ifluid,istop,muratio,ksteady,
     +  t,icrap, ineg,vinf,dxdt,sor,htemp,vrotate)
        dimension x(nn+1,nj), h(nn+1,2), tau(nn+1,2), vtan(nn+1,2)
        dimension f(nn+1), aa(nn+1), bb(nn+1),dxdt(nn+1,nj)
        dimension dd(nn+1), work1(4), work2(4), work3(3),htemp(nn+1,2)
        dimension u(nn+1,nj,2),sig(4),temp1(4,4), temp2(4,4),temp3(4)
        external funcs
        real muf,muratio
        imod = 0
        nsteps = 1
        deltattemp = deltat
        deltatperm = deltat
        do i = 1, nn+1
                 htemp(i,1) = h(i,1)
                 htemp(i,2) = h(i,2)
        enddo

        if ((deltattemp .gt. 0.10) .and. (vinf .gt. vrotate)) then
             print *, 'Invoking stepwise solution for dfluid.'
             imod = 1
```

```
            deltattest = deltatperm
            nsteps = 0
 51         nsteps = nsteps + 1
            deltattest = deltattest - 0.05
            if (deltattest .ge. 0.05) then
                go to 51
            endif
            print *, 'Number of steps required: ', nsteps
            deltattemp = 0.05
        endif



        do 6 l = 1, nsteps
        if (imod .eq. 1 .and. l .eq. nsteps) then
            deltattemp = deltatperm-(float(nsteps-1)*0.05)
        endif
        If (imod .eq. 1 .and. l .eq. nsteps) then
        print *, 'Temporary time interval: ', deltattemp
        endif
        iter = 0
        icrap = 0
        itermax = 20
        muf = visc(air) * muratio
        eps = 1.0*(10.**(-5.))
        eps2= 1.0*(10.**(-8.))
        rho = 1.23
        Print *, 'Working on the deicing fluid layer.'
 15     do 5 i = ifluid+1, istop-1
            f(i) = 0.0
 5      continue
c-----------------------------------------------------------------------
c       this part sets the iterations within a various timestep
c       initially equal to the previous value for h
c-----------------------------------------------------------------------
c-----------------------------------------------------------------------
c       for this particular time step, is the largest residual less than
c       the allowed error?
c-----------------------------------------------------------------------
        call calcf(f, tau, vtan, muf, deltattemp, rho, h, x, k,
     +  nn,nt,nj,ifluid,
     +  istop,dxdt)
        fmax = 0.0
        do 20 i = ifluid+1,istop-1
        if (abs(f(i)) .ge. fmax) then
            fmax = abs(f(i))
            imax = i
        endif
 20     continue
        if (fmax .le. eps .and. l .eq. nsteps) then
            print*, 'Number of iterations:', iter
            print*, 'fmax:',fmax
        endif
        if (fmax .le. eps) then
            go to 1000
```

```
        endif
c-----------------------------------------------------------------
c       here, the  magnitude of the error is greater than the allowed
c       error and we have to correct it by adding dh
c-----------------------------------------------------------------
        call calcj(bb,aa,dd,tau,vtan,muf,deltattemp,rho,h,x,k,
     +  nn,nt,nj,ifluid,
     +  istop,dxdt)
c-----------------------------------------------------------------
c       this part calculates the Jacobian
c
c       now, we need to solve J dh = -F
c-----------------------------------------------------------------
        do 30 i = ifluid+1, istop-1
        f(i) = -f(i)
 30     continue
        call sy(ifluid+1,istop-1,bb,dd,aa,f,nn+1)
c-----------------------------------------------------------------
c       solution returned in f
c-----------------------------------------------------------------
        do 50 i = ifluid+1, istop-1
        h(i,2) = h(i,2) + f(i)
        if (h(i,2) .lt. 0.) then
             h(i,2) = -h(i,2)
        endif

 50     continue
c-----------------------------------------------------------------
c       now predict the h(nn+1,2) and h(1, 2)
c-----------------------------------------------------------------

        call fluidstarter(ifluid,istop,nn,nj,h,x,rho,vtan,muf,tau,
     +  deltattemp,dxdt,icrap)
        if (h(istop,2) .lt. 0.) then
             h(istop,2) = -h(istop,2)
        endif

        iter = iter + 1
        if (iter .eq. itermax) then
        icrap = 1
        endif

        if (iter .eq. itermax .and. 1 .eq. nsteps) then
        print *, 'Maximum number of iterations reached in the'
        print *, 'deicing fluid module.  Moving on.'
        print *, 'Maximum residual in f:', fmax, ' at ', imax
        endif


 1000   do 150 i = ifluid, istop
        dx = x(i,1)-x(i-1,1)
        dpdx = -rho*(((vtan(i,2)-vtan(i-1,2))/dx)*(vtan(i,2))+dxdt(i,1)) -
     +       rho*((vtan(i,2)-vtan(i,1))/deltattemp)

        u(i,1, 2) = -(h(i,2)**2.)*(dpdx/(2.*muf)) + ((h(i,2)/muf)*
```

```
     +   (tau(i,2)))
         if (k .eq. ksteady) then
              h(i,2) = h(i,1)
         endif
 150     continue

c-----------------------------------------------------------------
         open (unit=27, file = 'houttemp.dat', status = 'unknown')
         write (27,*)'Time:', t
         write (27,*)'Vinf:', vinf
         write (27,*)'deltat:',deltattemp
         write (27,*)'Relaxation factor: ',sor
         write (27,*)'Maximum residual in f:', fmax, ' at ', imax
         write (27,*) 'fmax: ', fmax,' at ', imax
         write (27,46) 'i', 'h1','h2','x','dhdt','us','tau','vtan'
         do i = ifluid, nn+1
         write(27,45) i, h(i,1), h(i,2), x(i,1), (h(i,2)-
h(i,1))/deltattemp,
     +   u(i,1,2),tau(i,2),vtan(i,2)
         enddo
 45      format(1x,i3,3x,7(E12.5,3x))
 46      format(1x,8(3x,A12))
         close(27)
c-----------------------------------------------------------------
         if (icrap .eq. 1 .and. 1 .eq. nsteps) go to 47
         if (icrap .eq. 1 .and. 1 .ne. nsteps) go to 85
         if (fmax .ge. eps) go to 15
 85      if (1 .ne. nsteps)then
              do i = 1, nn+1
                   h(i,1) = h(i,2)
              enddo
         endif
 6       continue
 47      do i = i, nn+1
              h(i,1) = htemp(i,1)
         enddo
         deltat = deltatperm
c-----------------------------------------------------------------
         open (unit=27, file = 'houtperm.dat', status = 'unknown')
         write (27,*)'Time:', t
         write (27,*)'Vinf:', vinf
         write (27,*)'deltat:',deltat
         write (27,*)'Relaxation factor: ',sor
         write (27,*)'Maximum residual in f:', fmax, ' at ', imax
         write (27,46) 'i', 'h1','h2','x','dhdt','us','tau','vtan'
         do i = ifluid, nn+1
         write(27,45) i, h(i,1), h(i,2), x(i,1), (h(i,2)-h(i,1))/deltat,
     +   u(i,1,2),tau(i,2),vtan(i,2)
         enddo
         close(27)
c-----------------------------------------------------------------
         return
         end

c-----------------------------------------------------------------
```

```
c-----------------------------------------------------------------
c-----------------------------------------------------------------
      subroutine calcf(f, tau, vtan, muf, deltat, rho, h, x, k, nn,
     +  nt,nj,ifluid,
     +  istop,dxdt)
      dimension f(nn+1), tau(nn+1,2), vtan(nn+1,2), h(nn+1,2)
      dimension x(nn+1,nj),dxdt(nn+1,nj)
      real muf,isignh,isignhp,isignhm
c-----------------------------------------------------------------
c     this part calculates the f terms
c-----------------------------------------------------------------
      do 20 i = ifluid+1, istop-1

      if (h(i,2) .ge. 0.) isignh = 1.
c     if (h(i,2) .lt. 0.) isignh = -1.

      dx = x(i,1)-x(i-1,1)
      al = (x(i+1,1)-x(i,1))/dx

      zi  = (tau(i,2)*((h(i,2))**2.))/(2.*muf)+(rho*
     +  (((vtan(i,2)-vtan(i,1))/deltat)*((h(i,2))**3.))/(3.*muf))
      zip = ((tau(i+1,2)*((h(i+1,2))**2.))/(2.*muf))+(rho*
     +  (((vtan(i+1,2)-vtan(i+1,1))/deltat)*((h(i+1,2))**3.))/
     +  (3.*muf))
      zim = ((tau(i-1,2)*((h(i-1,2))**2.))/(2.*muf))+(rho*
     +  (((vtan(i-1,2)-vtan(i-1,1))/deltat)*((h(i-1,2))**3.))/
     +  (3.*muf))


      dvdx = (vtan(i+1,2) + (((al**2.)-1.)*vtan(i,2)) - (al**2.) *
     +  vtan(i-1,2))/(al*(al+1.)*dx)
      dvdxp = (vtan(i+1,2)-vtan(i,2)) / (al*dx)
      dvdxm = (vtan(i,2)-vtan(i-1,2)) / (dx)

      zi = zi + rho*(((vtan(i,2)+dxdt(i,1))*
     +  dvdx*(h(i,2)**3.))/(3.*muf))
      zip = zip + rho*(((vtan(i+1,2)+dxdt(i+1,1))*
     +  dvdxp*(h(i+1,2)**3.))/(3.*muf))
      zim = zim + rho*(((vtan(i-1,2)+dxdt(i-1,1))*
     +  dvdxm*(h(i-1,2)**3.))/(3.*muf))

      dhdx = (h(i+1,2) + (((al**2.)-1.)*h(i,2)) - (al**2.) *
     +  h(i-1,2))/(al*(al+1.)*dx)

      f(i) = (isignh*((h(i,2) - h(i,1))/deltat)) +
     +  ((zip + (((al**2.) - 1.)*zi) - ((al**2.)*zim))/
     +  (al*(al+1.)*dx)) + dxdt(i,1)*dhdx
 20   continue
      return
      end
c-----------------------------------------------------------------
c-----------------------------------------------------------------
c-----------------------------------------------------------------
      subroutine calcj(bb,aa,dd,tau,vtan,muf,deltat,rho,h,x,k,nn,nt,nj,
     +  ifluid, istop,dxdt)
```

```
      dimension tau(nn+1,2), vtan(nn+1,2), h(nn+1,2),aa(nn+1)
      dimension dd(nn+1),bb(nn+1)
      dimension x(nn+1,nj),dxdt(nn+1,nj)
      real muf,isignh
c-----------------------------------------------------------------
c     this part calculates the Jacobian terms
c-----------------------------------------------------------------
      do 30 i = ifluid+1, istop-1

      if (h(i,2) .ge. 0.) isignh = 1.
c     if (h(i,2) .lt. 0.) isignh = -1.

      dx = x(i,1)-x(i-1,1)
      al = (x(i+1,1)-x(i,1))/dx

      dvdx = (vtan(i+1,2) + (((al**2.)-1.)*vtan(i,2)) - (al**2.) *
     +  vtan(i-1,2))/(al*(al+1.)*dx)
      dvdxp = (vtan(i+1,2)-vtan(i,2)) / (al*dx)
      dvdxm = (vtan(i,2)-vtan(i-1,2)) / (dx)

      aa(i) = ((tau(i-1,2)*((h(i-1,2))**1.))/(1.*muf))+(rho*
     +  (((vtan(i-1,2)-vtan(i-1,1))/deltat)*((h(i-1,2))**2.))/
     +  (1.*muf))
      aa(i) = aa(i) + (rho*(((vtan(i-1,2)+dxdt(i-1,1))*
     +  dvdxm*(h(i-1,2)**2.))/(1.*muf)))
      aa(i) = aa(i) + dxdt(i,1)
      aa(i) = aa(i) * (1./(al*(al+1.)*dx))

      bb(i) = ((tau(i+1,2)*((h(i+1,2))**1.))/(1.*muf))+(rho*
     +  (((vtan(i+1,2)-vtan(i+1,1))/deltat)*((h(i+1,2))**2.))/
     +  (1.*muf))
      bb(i) = bb(i) + (rho*(((vtan(i-1,2)+dxdt(i-1,1))*
     +  dvdxm*(h(i-1,2)**2.))/(1.*muf)))
      bb(i) = bb(i) + dxdt(i,1)
      bb(i) = bb(i) * (-(al**2.)/(al*(al+1.)*dx))

      dd(i) = (tau(i,2)*((h(i,2))**1.))/(1.*muf)+(rho*
     +  (((vtan(i,2)-vtan(i,1))/deltat)*((h(i,2))**2.))/(1.*muf))
      dd(i) = dd(i) + (rho*(((vtan(i,2)+dxdt(i,1))*
     +  dvdx*(h(i,2)**2.))/(1.*muf)))
      dd(i) = dd(i) + dxdt(i,1)
      dd(i) = dd(i) * (((al**2.)-1.)/(al*(al+1.)*dx)) + (isignh/deltat)

 30   continue
      return
      end
c-----------------------------------------------------------------
c-----------------------------------------------------------------
      subroutine fluidstarter(ifluid,istop,nn,nj,h,x,rho,vtan,muf,tau,
     +  deltat,dxdt,icrap)
      real muf,rho,jacob
      real h(nn+1,2),vtan(nn+1,2),tau(nn+1,2),x(nn+1,nj),dxdt(nn+1,nj)
      real isignistop,isignistopm
      eps = 1.0*(10.**(-5.))
      iterstart = 0
```

```
        iterstartmax = 50
        icrap = 0
c-----------------------------------------------------------------------
c       this portion is used to determine h(ifluid,2) and
c       h(istop,2) based on a one-way explicit approximation
c       to the fluid equation
c-----------------------------------------------------------------------
        h(ifluid,2) = 0.

        dx = x(istop,1) - x(istop-1,1)
        al = (x(istop-1,1) - x(istop-2,1))/dx

 45     if (h(istop,2) .ge. 0.) isignistop = 1.
        if (h(istop,2) .lt. 0.) isignistop = -1.
        dpdxistop = (rho*(((dxdt(istop,1) + vtan(istop,2)) *
     +  ((-vtan(-1 + istop,2) + vtan(istop,2))/(al*dx))) +
     +  ((-vtan(istop,1) + vtan(istop,2))/deltat)))/(3.*muf)

        dpdxistopm = (rho*(((-vtan(-1 + istop,1) +
     +  vtan(-1 + istop,2))/deltat) +
     +  ((dxdt(-1 + istop,1) + vtan(-1 + istop,2))*
     +  ((-((al**2.)*vtan(-2 + istop,2)) +
     +  (-1. + (al**2.))*vtan(-1 + istop,2) + vtan(istop,2))/
     +  (al*(1. + al)*dx)))))/(3.*muf)

        tauistop = tau(istop,2)/(2.*muf)
        tauistopm = tau(istop-1,2)/(2.*muf)

        dhdx = (h(istop,2) - h(istop-1,2)) / (al*dx)


        resid = (isignistop*(h(istop,2)-h(istop,1)))/deltat
        resid = resid + (dpdxistop/(al*dx))*(h(istop,2)**3.)
        resid = resid - (dpdxistopm/(al*dx))*(h(istop-1,2)**3.)
        resid = resid + (tauistop/(al*dx))*(h(istop,2)**2.)
        resid = resid - (tauistop/(al*dx))*(h(istop-1,2)**2.)

        resid = resid + (dhdx * dxdt(istop,1))
        if (abs(resid) .le. eps) then
             go to 100
        endif
        jacob = (isignistop)/deltat
        jacob = jacob + 3.*(dpdxistop/(al*dx))*(h(istop,2)**2.)
        jacob = jacob + 2.*(tauistop/(al*dx))*(h(istop,2))
        jacob = jacob + (dxdt(istop,1)/(al*dx))
        dh = -resid/jacob
        if (h(istop,2) .lt. 0. .and. dh .lt. 0.) then
             dh = -dh
        endif
        h(istop,2) = h(istop,2) + dh
        iterstart = iterstart + 1
        if (iterstart .gt. iterstartmax) then
             return
        endif
        go to 45
```

```
100     return
        end


        subroutine hwriter(nn,xbar,t,h,ifluid)
        dimension xbar(nn),h(nn+1,2)
        do i = ifluid+1,nn+1
        write(72,400)t,xbar(i-1),h(i,1)
        enddo
400     format(1x, 3(E12.5,2x))
        return
        end

        subroutine tautopwriter(nn,xbar,t,tau,istag)
        dimension xbar(nn),tau(nn+1,2)
        do i = istag+1,nn+1
        write(73,400)t,xbar(i-1),tau(i,1)
        enddo
400     format(1x, 3(E12.5,2x))
        return
        end

        subroutine deltatopwriter(nn,xbar,t,delta,istag)
        dimension xbar(nn),delta(nn+1,2)
        do i = istag+1,nn+1
        write(74,400)t,xbar(i-1),delta(i,1)
        enddo
400     format(1x, 3(E12.5,2x))
        return
        end

        subroutine delta2topwriter(nn,xbar,t,delta2,istag)
        dimension xbar(nn),delta2(nn+1,2)
        do i = istag+1,nn+1
        write(75,400)t,xbar(i-1),delta2(i,1)
        enddo
400     format(1x, 3(E12.5,2x))
        return
        end

        subroutine delta3topwriter(nn,xbar,t,delta3,istag)
        dimension xbar(nn),delta3(nn+1,2)
        do i = istag+1,nn+1
        write(76,400)t,xbar(i-1),delta3(i,1)
        enddo
400     format(1x, 3(E12.5,2x))
        return
        end

        subroutine cptopwriter(nn,xbar,t,cp,istag)
        dimension xbar(nn),cp(nn)
        do i = istag+1,nn+1
        write(77,400)t,xbar(i-1),cp(i-1)
        enddo
400     format(1x, 3(E12.5,2x))
```

```
      return
      end

      subroutine taubotwriter(nn,xbar,t,tau,istag)
      dimension xbar(nn),tau(nn+1,2)
      do i = istag,1,-1
      write(83,400)t,xbar(i),tau(i,1)
      enddo
400   format(1x, 3(E12.5,2x))
      return
      end

      subroutine deltabotwriter(nn,xbar,t,delta,istag)
      dimension xbar(nn),delta(nn+1,2)
      do i = istag,1,-1
      write(84,400)t,xbar(i),delta(i,1)
      enddo
400   format(1x, 3(E12.5,2x))
      return
      end

      subroutine delta2botwriter(nn,xbar,t,delta2,istag)
      dimension xbar(nn),delta2(nn+1,2)
      do i = istag,1,-1
      write(85,400)t,xbar(i),delta2(i,1)
      enddo
400   format(1x, 3(E12.5,2x))
      return
      end

      subroutine delta3botwriter(nn,xbar,t,delta3,istag)
      dimension xbar(nn),delta3(nn+1,2)
      do i = istag,1,-1
      write(86,400)t,xbar(i),delta3(i,1)
      enddo
400   format(1x, 3(E12.5,2x))
      return
      end

      subroutine cpbotwriter(nn,xbar,t,cp,istag)
      dimension xbar(nn),cp(nn)
      do i = istag,1,-1
      write(87,400)t,xbar(i),cp(i)
      enddo
400   format(1x, 3(E12.5,2x))
      return
      end
      subroutine findh(hxbyc,xbar,y,nn,ifluid,h,ho,nt,nl)
      dimension xbar(nn), y(nn+1),h(nn+1,2)

      do 30 i=nl+1, nn
      if (xbar(i) .ge. hxbyc) then
           ifluid = i
           go to 40
      endif
```

```
 30      continue
 40      print *, 'Fluid begins at station: ', ifluid

         do 100 i = ifluid+1, nn+1
         h(i,1) = ho
 100     continue
c        h(nn+1,1) = -ho/10.
         do i = ifluid, nn+1
              h(i,2) = h(i,1)
         enddo
         return
         end

         subroutine findstag(nn, x, y, vtanp, rlen, istag, B)
         dimension x(nn+1), y(nn+1), vtanp(nn), rlen(nn)
         real B
c-----------------------------------------------------------------
c        search until we find a change in sign of the tangential velocity
c-----------------------------------------------------------------
         do 15 i = 1, nn
         if (vtanp(i) .ge. 0.0) then
              istag = i
              go to 10
         endif
 15      continue
c-----------------------------------------------------------------
c        we are assured that we have found the stagnation point
c
c        now we are looking for the slope, B
c-----------------------------------------------------------------
 10      B = (2.*(vtanp(istag) - vtanp(istag-1)))/((rlen(istag) +
     +   rlen(istag-1)))
         Print *, 'Stagnation at station: ', istag
c        Print *, 'Ratio of velocities: ', abs(vtanp(istag)/vtanp(istag-1))
         return
         end
         subroutine findstart(eta,fdp,acc)
         dimension ystart(3), dydx(3)
         external derivs
         external rkqc

c        print*, 'Finding the starting point for this boundary layer.'
         root1 = 1.23259
c        boundary layer guess at firstguess
         firstguess = 2.0
         itermax = 1000
         tol= 1.*(10.**(-5.))
         xs = 0.0
 30      iter = 0
         xend = firstguess
 50      iter = iter +1
         ystart(1) = 0.
         ystart(2) = 0.
         ystart(3) = root1
         h1 = xend/50.
```

```
        hmin = eps

        if (iter .eq. itermax) then
        print *, 'Maximum number of iterations. '
        go to 80
        endif

        call ODEINT(YSTART,3,xs,xend,tol,H1,HMIN,NOK,NBAD,DERIVS,
     +       RKQC,acc)
c       print *, iter, root1, xend, ystart(2), ystart(3)

        if (ystart(2) .ge. 0.99 .and. abs(ystart(3)) .le. 2.5E-02) then
c            print *, 'End of the Boundary Layer at eta = ', xend
        eta = xend
c            print *, 'With an initial guess of', root1, ' for ydp'
             fdp = root1
             go to 80
        endif
        if (ystart(2) .ge. 1.00 .and. ystart(3) .ge. 0.) then
             root1 = root1*0.995
             xend = 0.995 * xend
             go to 50
        endif

        if (ystart(2) .ge. 0.99 .and. ystart(3) .ge. 0.) then
             xend = xend * 1.006
             go to 50
        endif

        if (ystart(2) .le. 0.99 .and. ystart(3) .le. 0.) then
             root1 = root1*1.006
             xend =  xend *0.993
             go to 50
        endif


        if (ystart(2) .le. 0.99 .and. ystart(3) .ge. 0.) then
             xend = xend*1.006
             go to 50
        endif

        firstguess = .99 * firstguess
        go to 30
 80     return
        end


c-----------------------------------------------------------------
c       functions that are used in the deicing program
c-----------------------------------------------------------------
        subroutine freestream(vinf,t,isteady,tsteady,vinf0,vrotate,
     +                  vinfdot,trotate)
        if (isteady .eq. 0) then
             vinf = vinf0
             return
```

```
        endif
        vinf = (t-tsteady)*vinfdot + vinf0
        if (vinf .gt. vrotate .and. trotate .eq. 0) then
             trotate = t
        endif
c       if (vinf .ge. vrotate) then
c            vinf = vrotate
c       endif
        return
        end

        subroutine angleofattack(attack,alpha0,alphadot,t,trotate)
        if (trotate .eq. 0.) then
        attack = alpha0
        return
        endif
        attack = alphadot*(t-trotate) + alpha0
        return
        end
c-----------------------------------------------------------------
c       function angle(t, IGF)
c-----------------------------------------------------------------
c       function angle(t, IGF)
c-----------------------------------------------------------------
c       here, angle is the angle of attack in degrees as a function of
time
c-----------------------------------------------------------------
c       if (IGF .eq. 2) then
c            angle = 0.0
c            return
c       endif
c       angle = 10.0
c-----------------------------------------------------------------
c       keep it constant for now, but a angle of attack profile can
c       be determined at a later date
c-----------------------------------------------------------------
c       return
c       end

c-----------------------------------------------------------------
c       function rho(t)
c-----------------------------------------------------------------
        function rho(t)
c-----------------------------------------------------------------
c       here, rho is the fluid density in kg/m^3
c-----------------------------------------------------------------
        rho = 1.23
c-----------------------------------------------------------------
c       keep it constant for now
c-----------------------------------------------------------------
        return
        end

c-----------------------------------------------------------------
c       function visc(t)
```

```
c-----------------------------------------------------------------
        function visc(t)
        real nu, rho
c-----------------------------------------------------------------
c       here, mu is the air fluid dynamic viscosity in N s/m^2
c-----------------------------------------------------------------
        visc = rho(air)*nu(air)
c-----------------------------------------------------------------
c       keep it constant for now
c-----------------------------------------------------------------
        return
        end

c-----------------------------------------------------------------
c       function chord(t)
c-----------------------------------------------------------------
        function chord(t)
c-----------------------------------------------------------------
c       here, chord is chord length of the airfoil
c-----------------------------------------------------------------
        chord = 1.0
c-----------------------------------------------------------------
c       keep it constant for now
c-----------------------------------------------------------------
        return
        end


c-----------------------------------------------------------------
c       function Rey(t)
c-----------------------------------------------------------------
        function Rey(t)
c-----------------------------------------------------------------
c       here, Rey is the Reynolds number based on chord length
c-----------------------------------------------------------------
        Rey = (freestream2(0,t)*rho(t)*chord(t))/visc(t)
        return
        end
c-----------------------------------------------------------------
c       function freestream2(x,t)
c-----------------------------------------------------------------
        function freestream2(x,t)
c-----------------------------------------------------------------
c       here the final velocity is the takeoff velocity in m/s
c       and totaltime is the time it takes to reach that velocity in
seconds
c-----------------------------------------------------------------
        vfinal = 70.0 * (0.5144)
c-----------------------------------------------------------------
c       the first term is speed in knots, while the second term is
c       a conversion factor
c-----------------------------------------------------------------
        totaltime = 25.0
        acceleration = vfinal/totaltime
        v = acceleration * t
```

```
        freestream2 = v
        return
        end

c-----------------------------------------------------------------
c       function nu(air)
c
c       kinematic viscosity of the air
c-----------------------------------------------------------------
        real function nu(air)
        nu = 1.458e-05
        return
        end

c-----------------------------------------------------------------
c       function Rey(t)
c-----------------------------------------------------------------
        function Rey2(x,t)
c-----------------------------------------------------------------
c       here, Rey is the Reynolds number based on chord length
c-----------------------------------------------------------------
        Rey = (freestream2(0,t)*rho(t)*chord(t))/visc(t)
        return
        end

        real function alog(x)
        alog = dlog(x)
        return
        end

c-----------------------------------------------------------------
c       subroutine geominput
c
c       this routine will create the input specifications
c       it includes the NACA 4-digit number, the number of nodes
c       on the top and bottom of the airfoil.
c-----------------------------------------------------------------
        subroutine geominput(title,nl,nu,nn,nacanumber,IGF,AR)
        character * 40, title
c-----------------------------------------------------------------
c       this part gets the NACA number
c-----------------------------------------------------------------
 3      print 100
        read (*,103) title
 5      print 101
        read *, ni
 7      if (ni .eq. 1) then
             IGF = 2
             print 102
             read *, AR
             if (AR .le. 0.0) then
                 go to 7
             endif
             elseif (ni .eq. 2) then
                 IGF = 1
```

```
        endif
        if (IGF .eq. 0) go to 5
        if (IGF .eq. 2) go to 90

 10     print *, 'Please input four digit NACA number'
        read *, nacanumber
        if (nacanumber .gt. 9999 .or. nacanumber .le. 0) then
                go to 10
        endif
c-----------------------------------------------------------------
c       getting number of nodes on top and bottom surfaces
c-----------------------------------------------------------------
 90     print *, 'Please input the number of nodes on upper surface.'
        read *, nu
        print *, 'Please input the number of nodes on lower surface.'
        read *, nl
        if (nu+nl .gt. 1000) then
                print *, 'Too many nodes...'
                go to 90
        elseif (nu .le. 1 .or. nl .le. 1) then
                print *, 'Need more nodes...'
                go to 90
        endif

        nn = nl + nu
        open (unit = 7, file = 'xy.dat', status = 'unknown')
        write(7,103) title
        write(7,600) nl
        write(7,600) nu
        write(7,600) nn
        write(7,600) IGF
        close(unit = 7)
 600    format(1x, i5)
 700    format(1x, f10.4)
 100    format(' Input the title for this run.')
 101    format(' Is this geometry for an ellipse (1) Yes (2) No ?',$)
 102    format(' What is the ratio of the major axis to the minor
axis?',$)
 103    format(A)
        return
        end
        subroutine getnn(nu,nl,nn, title, IGF)
        character *40, title
        open (unit = 7, file = 'xy.dat', status = 'old')
        rewind (unit =7)
        read(7,650) title
        read(7,600) nl
        read(7,600) nu
        read(7,600) nn
        read(7,600) IGF
        close (unit = 7)
 600    format(1x, i5)
 650    format(A)
 700    format(1x, f10.4)
        return
```

```
      end
c-----------------------------------------------------------------
c     djcronin@iastate.edu
c
c     this proram creates the airfoil geometry
c
c-----------------------------------------------------------------
      subroutine grid(nacanumber,nu,nl,nn,x,y,title,IGF)
      dimension x(nn+1),y(nn+1)
      character *40, title
      pi = atan(1.)*4.
      ieps = nacanumber/1000
      iptmax = nacanumber/100 - 10*ieps
      itau = nacanumber - 1000 * ieps - 100 * iptmax
      epsmax = ieps * 0.01
      ptmax = iptmax * 0.1
      tau = itau * 0.01
c-----------------------------------------------------------------
c     here the chord length is set to unity and all work
c     is done non-dimensionally
c
c
c     setting up the node points for the naca airfoil
c-----------------------------------------------------------------
      npoints = nl
      sign = -1.0
      nstart = 0
      do 110 nsurf = 1,2
      do 100 n = 1, npoints
      fract = float(n-1)/float(npoints)
      z = .5 * (1. - cos(pi*fract))
      i = nstart + n
      call body(z,sign,x(i),y(i),nacanumber,epsmax,ptmax,tau)
 100  continue
      npoints = nu
      sign = 1.0
      nstart = nl
 110  continue
      nodtot = nu + nl
      x(nodtot+1) = x(1)
      y(nodtot+1) = y(1)
      print *, 'Geometry for airfoil complete.'
      open (unit = 7, file = 'xy.dat', status = 'old')
c-----------------------------------------------------------------
c     write to the file 'xy.dat'
c-----------------------------------------------------------------
      kount = 0
      write(7,103) title
      write(7,601) nl
      write(7,601) nu
      write(7,601) nn
      write(7,601) IGF
 103  format(A)
 601  format(1x, i5)
      do 510 i = 1, nodtot+1
```

```
      kount = kount+1
      write (7,600) x(i),y(i)
 510  continue
      print *, 'Data written to file xy.dat.'
      close (unit = 7)
 600  format(1x,2(3x,f12.7))
 700  format(1x, f10.4)

      end
c-----------------------------------------------------------------
c     subrountine body
c     return coordinates of point on a body surface
c
c     z = node spacing parameter
c     x,y = cartesian coordinates
c     sign = +1 for upper surface
c          = -1 for lower surface
c
c-----------------------------------------------------------------
      subroutine body(z,sign,x,y,nacanumber,epsmax,ptmax,tau)
      if (sign .lt. 0.0) z = 1. - z
      call naca45(z,thick,camber,beta,nacanumber,epsmax,ptmax,tau)
      x = z - sign*thick*sin(beta)
      y = camber + sign*thick*cos(beta)
      return
      end
c-----------------------------------------------------------------
c     subrountine naca45
c
c     evaluate thickness and camber for NACA 4- or 5-digit airfoil
c-----------------------------------------------------------------
      subroutine naca45(z,thick,camber,beta,nacanumber,epsmax,ptmax,tau)
      thick = 0.0
      if (z .lt. 1.e-10)    go to 100
      thick = 5. *tau* (.2969*sqrt(z) - z*(0.126 + z*(.3537
     +    - z *(.2843 - z * 0.1015))))
 100  if (epsmax .eq. 0.0)        go to 130
      if (nacanumber .gt. 9999)   go to 140
      if (z .gt. ptmax)           go to 110
      camber = epsmax/ptmax/ptmax*(2.*ptmax -z)*z
      dcamdx = 2.* epsmax/ptmax/ptmax*(ptmax -z)
      go to 120
 110  camber = epsmax/(1. - ptmax)**2*(1. + z- 2*ptmax)*(1.-z)
      dcamdx = 2.*epsmax/(1.-ptmax)**2*(ptmax -z)
 120  beta = atan(dcamdx)
      return
 130  camber = 0.0
      beta = 0.0
      return
 140  if (z .gt. ptmax)     go to 150
      w = z/ptmax
      camber = epsmax*w*((w- 3.)*w+3.-ptmax)
      dcamdx = epsmax*3.*w*(1.-w)/ptmax
      go to 120
 150  camber = epsmax*(1. -z)
```

```
      dcamdx = -epsmax
      go to 120
      end

      subroutine helper(nn,nj,ifluid,istop,x,vtan,t,h,deltat,muratio,
     +  tau)
c-----------------------------------------------------------------
c       this subroutine helps look at parameters in the
c       differential equation:
c       dh/dt + d/dx(((tau (h^2))/2 muf) - (dp/dx) (h^3))/3 muf)) = 0
c-----------------------------------------------------------------
      dimension x(nn+1,nj), vtan(nn+1,2), h(nn+1,2), tau(nn+1,2)
      real muf
      real muratio
      rho = 1.23
      muf = muratio * visc(air)
      open (unit=32, file='qhelper.dat', status = 'unknown')
      write(32,*) 'Time:', t
      write(32,*) 'deltat:', deltat
      write(32,*) ' '
      write(32,100) 'Position', 'x', 'vtan', 'tau', 'h1', 'h2',
     +  'hsquared', 'hcubed', 'dvdt', 'dvdx', 'tauterm',
     +  'dpdxterm', 'ddxtauterm','ddxdpdxterm', 'sum of last two','dhdt'
      do i = ifluid+1, istop-1
      dx = x(i,1) - x(i-1,1)
      al = (x(i+1,1) - x(i,1))/dx
      dhdt = (h(i,2)-h(i,1))/deltat
      hsquared = h(i,2)*h(i,2)
      hcubed = h(i,2)*h(i,2)*h(i,2)
      dvdt = (vtan(i,2)-vtan(i,1))/deltat
      dvdx = (vtan(i+1,2) + ((al**2.)-1.)*vtan(i,2) -
     +  (al**2.)*vtan(i-1,2))
     +  / (al*(al+1.)*dx)
      tauterm = (tau(i,2)*(h(i,2)**2.))/(2.*muf)
      dpdx = -rho*(dvdt + vtan(i,2)*dvdx)
      dpdxterm = (-dpdx*(h(i,2)**3.))/(3.*muf)
      tautermp = (tau(i+1,2)*(h(i,2)**2.))/(2.*muf)
      tautermm = (tau(i-1,2)*(h(i-1,2)**2.))/(2.*muf)
      dvdxp = (vtan(i+1,2) -vtan(i,2))/(al*dx)
      dvdxm = (vtan(i,2) - vtan(i-1,2))/(dx)
      dvdtp = (vtan(i+1,2) -vtan(i+1,1))/deltat
      dvdtm = (vtan(i-1,2) -vtan(i-1,1))/deltat
      dpdxp = -rho*(dvdtp + vtan(i+1,2)*dvdxp)
      dpdxm = -rho*(dvdtm + vtan(i-1,2)*dvdxm)
      dpdxtermp = (-dpdxp*(h(i+1,2)**3.))/(3.*muf)
      dpdxtermm = (-dpdxm*(h(i-1,2)**3.))/(3.*muf)
      ddxtauterm = (tautermp + ((al**2.)-1.)*tauterm
     +  -(al**2.)*tautermm)
     +  / (al*(al+1.)*dx)
      ddxdpdxterm = (dpdxtermp + ((al**2.)-1.)*dpdxterm
     +  -(al**2.)*dpdxtermm)
     +  / (al*(al+1.)*dx)
      write(32,101) i, x(i,1), vtan(i,2), tau(i,2), h(i,1), h(i,2),
     +  hsquared, hcubed, dvdt, dvdx, tauterm, dpdxterm, ddxtauterm,
     +  ddxdpdxterm, ddxtauterm + ddxdpdxterm, dhdt
```

```
        enddo
        close(32)
 100    format(1x, 16(a15,3x))
 101    format(1x,i5,5(3x, f12.5),2(3x, e12.5),2(3x, f12.5),6(3x,e12.5))
        return
        end
c-----------------------------------------------------------------
c       these are functions the grid generation program will need
c-----------------------------------------------------------------
        subroutine initialgrid(nn,nj,ratio,yftop,x,y,h,nt,k,istag,ifluid,
     +  yoldg,dydt,deltat)
        dimension x(nn+1,nj), y(nn+1,nj), h(nn+1,2),yoldg(nn+1,nj)
        dimension dydt(nn+1,nj)
c-----------------------------------------------------------------
c       here the plate length is set to   unity and all work
c       is done   non-dimensionally
c-----------------------------------------------------------------
c-----------------------------------------------------------------
c       next, we put the surface in at y(i,1)
c-----------------------------------------------------------------
        do 70 i   = ifluid, nn+1
        y(i,1) = h(i,2)
 70     continue

c-----------------------------------------------------------------
c       for each x station, we need to find the distance between h(x) and
c       yf to set the grid
c-----------------------------------------------------------------
        do 60 i   = istag, nn+1
             sum = 0.0
             do 100 j = 0, nj-2
             sum = sum + ratio**(j)
 100         continue
c            dy = (yftop-y(i,1))/sum
             dy = (yftop)/sum

        do 50 j   = 2, nj
        y(i,j) = y(i,j-1) + ratio**(j-2) * dy
 50     continue
 60     continue
        do i = 1, nn+1
        do j = 1, nj
        if (i .ne. istag .or. i .ne. istag-1 .or. i .eq. istag+1) then
        dydt(i,j) = (y(i,j) - yoldg(i,j))/deltat
        endif
        enddo
        enddo

        return
        end
c-----------------------------------------------------------------
c       subroutine liftdrag
c-----------------------------------------------------------------
        subroutine liftdrag (nn,cp,cl,cd,rlen,vinf,attack,theta,x,y,cm,
     +  isteady,tsteady,t,ksteady,k,vtan,clsteady)
```

```
      dimension cp(nn),rlen(nn),theta(nn),x(nn+1),y(nn+1)
      dimension vtan(nn)
      pi = atan(1.) * 4.
      cdold = cd
      clold = cl
      alpha = attack * (pi/180.)
      cfy = 0.0
      cfx = 0.0
      cm = 0.0
      do 20 i = 1, nn
           xmid = 0.5*(x(i) + x(i+1))
           ymid = 0.5*(y(i) + y(i+1))
           dx = x(i+1) - x(i)
           dy = y(i+1) - y(i)
           cfx = cfx + cp(i)*dy
           cfy = cfy - cp(i)*dx
           cm = cm + cp(i)*(dx*xmid + dy*ymid)
20    continue
      cl = cos(alpha)*cfy - sin(alpha)*cfx
      print *, 'Section lift coefficient: ', cl
      cd = cos(alpha)*cfx + sin(alpha)*cfy

      cfy = 0.0
      cfx = 0.0

      do 30 i = 1, nn
           xmid = 0.5*(x(i) + x(i+1))
           ymid = 0.5*(y(i) + y(i+1))
           dx = x(i+1) - x(i)
           dy = y(i+1) - y(i)
         temp = 1. - ((vtan(i)/vinf)**2.)
           cfx = cfx + temp*dy
           cfy = cfy - temp*dx
           cm = cm + temp*(dx*xmid + dy*ymid)
30    continue
      clsteady = cos(alpha)*cfy - sin(alpha)*cfx
      cdsteady = cos(alpha)*cfx + sin(alpha)*cfy
      if (k .eq. 1) then
      print *, 'Steady-state section lift coefficient: ', clsteady
      print *, 'Steady-state section drag coefficient: ', cdsteady
      endif

      epsl = abs(clold-cl)/abs(cl)
      epsd = abs(cdold-cd)/abs(cd)
      if (epsl .le. 0.005 .and. epsd .le. 0.005 .and. tsteady .eq. 0.0
     +  .and. isteady .eq. 0) then
      isteady = 1
      ksteady = k
      Print *, 'Flow is now steady.'
      tsteady = t
      endif
      if (abs(cl) .le. 0.00001 .and. k .gt. 1) then
      if (k-ksteady .ne. 0 .and. isteady .eq. 0)then
      isteady = 1
      tsteady = t
```

```
      ksteady = k
      print *, 'Turned it on here.'
      endif
      endif
      return

      end
      subroutine metric(x,y,xoldg,yoldg,dxdt,dydt,nn,nj,deltat)
      dimension x(nn+1,nj), y(nn+1,nj),xoldg(nn+1,nj),yoldg(nn+1,nj)
      dimension dxdt(nn+1,nj),dydt(nn+1,nj)

      do i = 1,nn+1
      do j = 1,nj
      dxdt(i,j) = (x(i,j)-xoldg(i,j))/deltat
      dydt(i,j) = (y(i,j)-yoldg(i,j))/deltat
      enddo
      enddo
      return
      end

      subroutine gridstart(x,y,xoldg,yoldg,nn,nj)
      dimension x(nn+1,nj), y(nn+1,nj),xoldg(nn+1,nj),yoldg(nn+1,nj)
      do i = 1,nn+1
      do j = 1,nj
      xoldg(i,j) = x(i,j)
      yoldg(i,j) = y(i,j)
      enddo
      enddo
      return
      end
      SUBROUTINE ODEINT(YSTART,NVAR,X1,X2,EPS,H1,HMIN,NOK,NBAD,DERIVS,RK
     *QC,acc)
      PARAMETER (MAXSTP=10000,NMAX=10,TWO=2.0,ZERO=0.0,TINY=1.E-30)
      COMMON /PATH/ KMAX,KOUNT,DXSAV,XP(200),YP(10,200)
      DIMENSION YSTART(NVAR),YSCAL(NMAX),Y(NMAX),DYDX(NMAX)
        external derivs
        external rkqc
      X=X1
      H=SIGN(H1,X2-X1)
      NOK=0
      NBAD=0
      KOUNT=0
      DO 11 I=1,NVAR
        Y(I)=YSTART(I)
11    CONTINUE
      XSAV=X-DXSAV*TWO
      DO 16 NSTP=1,MAXSTP
        CALL DERIVS(X,Y,DYDX,acc)
        DO 12 I=1,NVAR
          YSCAL(I)=ABS(Y(I))+ABS(H*DYDX(I))+TINY
12      CONTINUE
        IF(KMAX.GT.0)THEN
          IF(ABS(X-XSAV).GT.ABS(DXSAV)) THEN
            IF(KOUNT.LT.KMAX-1)THEN
              KOUNT=KOUNT+1
```

```
            XP(KOUNT)=X
            DO 13 I=1,NVAR
              YP(I,KOUNT)=Y(I)
13          CONTINUE
            XSAV=X
          ENDIF
        ENDIF
      ENDIF
      IF((X+H-X2)*(X+H-X1).GT.ZERO) H=X2-X
      CALL RKQC(Y,DYDX,NVAR,X,H,EPS,YSCAL,HDID,HNEXT,DERIVS,acc)
      IF(HDID.EQ.H)THEN
        NOK=NOK+1
      ELSE
        NBAD=NBAD+1
      ENDIF
      IF((X-X2)*(X2-X1).GE.ZERO)THEN
        DO 14 I=1,NVAR
          YSTART(I)=Y(I)
14      CONTINUE
        IF(KMAX.NE.0)THEN
          KOUNT=KOUNT+1
          XP(KOUNT)=X
          DO 15 I=1,NVAR
            YP(I,KOUNT)=Y(I)
15        CONTINUE
        ENDIF
        RETURN
      ENDIF
      IF(ABS(HNEXT).LT.HMIN) then !PAUSE 'Stepsize smaller than minimum.'
      endif
      H=HNEXT
16    CONTINUE
        ! PAUSE 'Too many steps.'
      RETURN
      END
      SUBROUTINE RK4(Y,DYDX,N,X,H,YOUT,DERIVS,acc)
      PARAMETER (NMAX=10)
      DIMENSION Y(N),DYDX(N),YOUT(N),YT(NMAX),DYT(NMAX),DYM(NMAX)
        external derivs
      HH=H*0.5
      H6=H/6.
      XH=X+HH
      DO 11 I=1,N
        YT(I)=Y(I)+HH*DYDX(I)
11    CONTINUE
      CALL DERIVS(XH,YT,DYT,acc)
      DO 12 I=1,N
        YT(I)=Y(I)+HH*DYT(I)
12    CONTINUE
      CALL DERIVS(XH,YT,DYM,acc)
      DO 13 I=1,N
        YT(I)=Y(I)+H*DYM(I)
        DYM(I)=DYT(I)+DYM(I)
13    CONTINUE
      CALL DERIVS(X+H,YT,DYT,acc)
```

```
      DO 14 I=1,N
        YOUT(I)=Y(I)+H6*(DYDX(I)+DYT(I)+2.*DYM(I))
14    CONTINUE
      RETURN
      END

        subroutine DERIVS(x,y,dydx,acc)
        real y(3),dydx(3)
        real x, acc
        dydx(1) = y(2)
        dydx(2) = y(3)
        dydx(3) = y(2)*y(2) - 1. - y(1)*y(3) - acc + (acc*y(2))
        return
        end
      SUBROUTINE RKQC(Y,DYDX,N,X,HTRY,EPS,YSCAL,HDID,HNEXT,DERIVS,acc)
      PARAMETER (NMAX=10,FCOR=.0666666667,
     *    ONE=1.,SAFETY=0.9,ERRCON=6.E-4)
        EXTERNAL DERIVS
      DIMENSION Y(N),DYDX(N),YSCAL(N),YTEMP(NMAX),YSAV(NMAX),DYSAV(NMAX)
      PGROW=-0.20
      PSHRNK=-0.25
      XSAV=X
      DO 11 I=1,N
        YSAV(I)=Y(I)
        DYSAV(I)=DYDX(I)
11    CONTINUE
      H=HTRY
1     HH=0.5*H
      CALL RK4(YSAV,DYSAV,N,XSAV,HH,YTEMP,DERIVS,acc)
      X=XSAV+HH
      CALL DERIVS(X,YTEMP,DYDX,acc)
      CALL RK4(YTEMP,DYDX,N,X,HH,Y,DERIVS,acc)
      X=XSAV+H
      IF(X.EQ.XSAV) then !PAUSE 'Stepsize not significant in RKQC.'
        endif
      CALL RK4(YSAV,DYSAV,N,XSAV,H,YTEMP,DERIVS,acc)
      ERRMAX=0.
      DO 12 I=1,N
        YTEMP(I)=Y(I)-YTEMP(I)
        ERRMAX=MAX(ERRMAX,ABS(YTEMP(I)/YSCAL(I)))
12    CONTINUE
      ERRMAX=ERRMAX/EPS
      IF(ERRMAX.GT.ONE) THEN
        H=SAFETY*H*(ERRMAX**PSHRNK)
        GOTO 1
      ELSE
        HDID=H
        IF(ERRMAX.GT.ERRCON)THEN
          HNEXT=SAFETY*H*(ERRMAX**PGROW)
        ELSE
          HNEXT=4.*H
        ENDIF
      ENDIF
      DO 13 I=1,N
        Y(I)=Y(I)+YTEMP(I)*FCOR
```

```
13    CONTINUE
      RETURN
      END
c-----------------------------------------------------------------
c       writing the output data to file
c
c       subroutine output
c-----------------------------------------------------------------
        subroutine output1(k, ni, nj, nt, deltat, x, tau, delta,
     +  delta2, delta3, shape,vtan,h,istart,istop,isign,u,t)
        dimension x(ni,nj), tau(ni,2), delta(ni,2), delta2(ni,2)
        dimension delta3(ni,2), shape(ni,2), vtan(ni,2),h(ni,2)
        dimension u(ni,nj,2)
        real nu
        character *20, filename
        if (isign .eq. 1) then
             filename = 'output.top.dat'
        endif
        if (isign .eq. -1) then
             filename = 'output.bot.dat'
        endif

        open (unit = 7, file = filename, status = 'unknown')
        write(7,659) t
        write(7,*) ' '
        write(7,700)

        do 510 i = istart+isign, istop,isign
        write(7,600) i,x(i,1),vtan(i,2),tau(i,2),delta(i,2),delta2(i,2),
     +       delta3(i,2), shape(i,2), isign*
     +       vtan(i,2)*delta2(i,2)/nu(air),
     +       h(i,2),1000.*(tau(i,2)/(0.5*rho(air)*(vtan(i,2)**2.))),
     +       u(i,1,2)
 510    continue
        if (isign .eq. 1) then
        print *, 'Data written to file output.top.dat.'
        endif
        if (isign .eq. -1) then
        print *, 'Data written to file output.bot.dat.'
        endif

        close (unit = 7)
 600    format(1x,i5,11(3x,f15.7))
 659    format(1x,'Time:','(','seconds',
     +           ')',2x,f12.4)
 700    format(1x,3x,'Element',5x,'x',10x,'vtan', 10x,'tau',10x,'delta',
     +  10x,
     +  'displacement thickness',
     +  10x,'momentum thickness',
     +  10x,'Shape Factor',10x,'Rey disp thick', 10x,
     +  'deicing fluid depth',10x,'skin friction coef*1000',
     +  10x,'Slip velocity')
        return
        end
```

```
c-----------------------------------------------------------------------
c       writing the output data to file
c
c       subroutine output
c-----------------------------------------------------------------------
        subroutine output2(k, ni, nj, nt, deltat, x, y, delta,u,v,
     +  vtan,tau,istart,istop,isign,t)
        dimension x(ni,nj), y(ni,nj), delta(ni,2)
        dimension u(ni,nj,2),tau(ni,2)
        dimension v(ni,nj,2), vtan(ni,2)
        character * 20, filename
        character *7, temp
        real nu
c-----------------------------------------------------------------------
c       find the stations
c-----------------------------------------------------------------------
        do 20 m = 1,4
        if (m .eq. 1 .and. isign .eq. 1) then
                temp = 'top.25'
        elseif (m .eq. 2 .and. isign .eq. 1) then
                temp = 'top.50'
        elseif (m .eq. 3 .and. isign .eq. 1) then
                temp = 'top.75'
        elseif (m .eq. 4 .and. isign .eq. 1) then
                temp = 'top.100'
        endif
        if (m .eq. 1 .and. isign .eq. -1) then
                temp = 'bot.25'
        elseif (m .eq. 2 .and. isign .eq. -1) then
                temp = 'bot.50'
        elseif (m .eq. 3 .and. isign .eq. -1) then
                temp = 'bot.75'
        elseif (m .eq. 4 .and. isign .eq. -1) then
                temp = 'bot.100'
        endif

        filename = 'st.'//temp//'.dat'

        do 30 i = istart+isign, istop, isign
        if (abs(x(i,1)) .ge. 0.249*float(m)) then
        iwrite = i
        go to 40
        endif
 30     continue
 40      i = iwrite
         open (unit = 7, file = filename, status = 'unknown')
         write(7,659) t
         write(7,660) x(i,1)
         write(7,661) vtan(i,2)
         write(7,662) delta(i,2)
         write(7,*) ' '
         if (tau(i,2) .gt. 0.) then
                ustar = (tau(i,2)/rho(air))**(1./2.)
```

```
      else
              write(7,*) 'Flow has separated at this point.'
              ustar = 0.
      endif

      write(7,700) 'Position', 'y'//temp, 'eta'//temp,
     +      'u'//temp, 'v'//temp, 'fprime'//temp, 'u+'//temp,
     +      'y+'//temp

      do 510 j = 1, nj
      if (ustar .ne. 0.) then
      write(7,600) j,y(i,j),((y(i,j)-y(i,1))/delta(i,2)),
     + u(i,j,2), v(i,j,2), (u(i,j,2)/vtan(i,2)),
     + isign*u(i,j,2)/ustar, (((y(i,j)-y(i,1))*ustar)/nu(air))
      else
      write(7,600) j,y(i,j),((y(i,j)-y(i,1))/delta(i,2)),
     + u(i,j,2), v(i,j,2), (u(i,j,2)/vtan(i,2)),
     + 999.,999.
      endif
510   continue
551   close (unit = 7)
20    continue
600   format(1x,i5,7(3x,f20.7))
659   format(1x,'Time:','(','seconds',
     +          ')',2x,f12.4)
660   format(1x, 'Information at x = ',2x, f12.4)
661   format(1x, 'Tangential velocity at this station: ',2x, f12.4)
662   format(1x, 'Boundary layer thickness at this station: ',2x,f12.4)
700   format(1x,8(6x,A))
      return
      end
c-----------------------------------------------------------------
c     subroutine potentials
c-----------------------------------------------------------------
c     calculates the potential function at a particular point
c     as a function of the sum of the potential due to the
c     freestream, the potential due to the sources and the potential
c     due to the vortices
c-----------------------------------------------------------------
      subroutine potentials(nn, q, xbar, ybar, rlen, phi,
     +  phis, phiv, theta, x, y,attack,vinf,phiold,k,circ,circx,circy,nt)
      dimension xbar(nn), ybar(nn), q(nn+1), rlen(nn), phi(nn)
      dimension phis(nn), phiv(nn),theta(nn), x(nn+1),y(nn+1)
      dimension phiold(nn),circ(nt), circx(nt,nt),circy(nt,nt)

      gamma = q(nn+1)
      pi = atan(1.) * 4.
      alpha = attack * (pi/180.)

      do 15 i = 1, nn
      phiold(i) = phi(i)
15    continue
      do 100 i = 1, nn
      phi(i) = vinf * xbar(i) *cos(alpha) + vinf * ybar(i) *sin(alpha)
100   continue
```

```
      do 200 i = 1, nn
      do 300 j = 1, nn
      xstar = (xbar(i) - x(j))  * cos(theta(j)) + (ybar(i) - y(j)) *
     +      sin(theta(j))
      ystar = (ybar(i) - y(j))  * cos(theta(j)) - (xbar(i) - x(j)) *
     +      sin(theta(j))

      phivj = vortex(gamma,ystar,(xstar-rlen(j))) -
     +      vortex(gamma,ystar,xstar)

      phisj = source(q(j),ystar,(xstar-rlen(j))) -
     +      source(q(j),ystar,xstar)

      phis(i) = phis(i) + phisj
      phiv(i) = phiv(i) + phivj

 300  continue
 200  continue

      do 400 i = 1, nn
      phi(i) = phi(i) + phis(i) + phiv(i)
 400  continue
c-----------------------------------------------------------------
c     this part of the calculation adds the potential at each node
c     due to the trialing vortices
c-----------------------------------------------------------------
      if (k .gt. 1) then
      do 50 m = 1, k-1
      do 60 i = 1, nn
      thetahat = atan2(ybar(i)-circy(m,k),xbar(i)-circx(m,k))
      phiadd = (-circ(m)/(2. * pi))*thetahat
      phi(i) = phi(i) + phiadd
 60   continue
 50   continue
      endif

      return
      end

      function source(q,y,z)
      real pi, z, y, q, source
      pi = atan(1.) * 4.
      if (y .eq. 0.0 .and. z .gt. 0.0) then
           rone = pi/2.
           elseif (y .eq. 0.0 .and. z .lt. 0.0) then
           rone = -pi/2.
           else
           rone = atan(z/y)
      endif
      source = (-q/(2* pi)) * (-z + y*rone + z*alog((z**2. +
     +      y**2.)**(1./2.)))
      return
      end
```

```
      function vortex(gamma,y,z)
      real pi, z, y, gamma, vortex
      pi = atan(1.) * 4.
      if (z .eq. 0.0 .and. y .gt. 0.0) then
           rone = pi/2.
           elseif (z .eq. 0.0 .and. y .lt. 0.0) then
           rone = -pi/2.
           else
           rone = atan(y/z)
      endif
      vortex = (gamma/(2* pi)) * (z*rone + y*alog((z**2. +
     +     y**2.)**(1./2.)))
      return
      end
c-----------------------------------------------------------------
c     writing the output data to file
c
c     subroutine output
c-----------------------------------------------------------------
      subroutine
potoutput(nn,nu,nl,attack,vinf,x,y,xbar,ybar,cp,cl,cd,cm,
     +          nacanumber,phi,phiold,q,t,k,circ,nt,IGF,vtan)
      dimension x(nn+1),y(nn+1),cp(nn),xbar(nn),ybar(nn),vtan(nn)
      dimension phi(nn),phiold(nn),q(nn+1),circ(nt)
      pi = atan(1.)*4.
      if (k .eq. 1) then
      call potoutput2(nn,nu,nl,attack,vinf,x,y,xbar,ybar,cp,cl,cd,cm,
     +          nacanumber,phi,phiold,q,t,k,circ,nt,IGF,vtan)
      endif
      open (unit = 7, file = 'potoutput.dat', status = 'unknown')
      if (IGF .eq. 1) write(7,650)nacanumber
      write(7,651)nl
      write(7,652)nu
      write(7,653)nn
      write(7,654)vinf
      write(7,655)attack
      write(7,659)t
      write(7,656)cl
      write(7,657)cd
      write(7,658)cm
      write(7,660)circ(k)
      write(7,*) ' '
      write(7,700)
      kount = 0
      do 510 i = 1, nn
      write (7,600) i,x(i),y(i),xbar(i),ybar(i),phi(i),phiold(i),
     +     q(i), cp(i), 1. - ((vtan(i)/vinf)**2.)
 510  continue
      print *, 'Data written to file potoutput.dat.'
      close (unit = 7)
 600  format(1x,i5,9(3x,f14.7))
 650  format(1x,'NACA number:',2x,i5)
 651  format(1x,'Number of nodes on lower surface:',2x,i5)
 652  format(1x,'Number of nodes on upper surface:',2x,i5)
 653  format(1x,'Total number of nodes:',2x,i5)
```

```
654     format(1x,'Free stream velocity:','(','units length/units time',
    +           ')',2x,f12.4)
655     format(1x,'Angle of Attack:','(','degrees',
    +           ')',2x,f12.4)
656     format(1x,'Section lift coefficient:',2x,f12.4)
657     format(1x,'Section drag coefficient:',2x,f12.4)
658     format(1x,'Section moment coefficient:',2x,f12.4)
659     format(1x,'Time:','(','seconds',
    +           ')',2x,f12.4)
660     format(1x,'Circulation:',2x,f12.4)
700     format(1x,3x,'Element',5x,'x',12x,'y',12x,'xbar',12x,'ybar',
    +   12x,'phi',12x,'phiold',12x,'q',12x,'Cp',12x,'Cpsteady')
        return
        end


c-----------------------------------------------------------------------
c       writing the output data to file
c
c       subroutine output
c-----------------------------------------------------------------------
        subroutine
potoutput2(nn,nu,nl,attack,vinf,x,y,xbar,ybar,cp,cl,cd,cm,
    +           nacanumber,phi,phiold,q,t,k,circ,nt,IGF,vtan)
        dimension x(nn+1),y(nn+1),cp(nn),xbar(nn),ybar(nn),vtan(nn)
        dimension phi(nn),phiold(nn),q(nn+1),circ(nt)
        pi = atan(1.)*4.
        open (unit = 7, file = 'potoutput.steady.dat', status = 'unknown')
        if (IGF .eq. 1) write(7,650)nacanumber
        write(7,651)nl
        write(7,652)nu
        write(7,653)nn
        write(7,654)vinf
        write(7,655)attack
        write(7,659)t
        write(7,656)cl
        write(7,657)cd
        write(7,658)cm
        write(7,660)circ(k)
        write(7,*) ' '
        write(7,700)
        kount = 0
        do 510 i = 1, nn
        write (7,600) i,x(i),y(i),xbar(i),ybar(i),phi(i),phiold(i),
    +       q(i), 1. - ((vtan(i)/vinf)**2.)
510     continue
        print *, 'Data written to file potoutput.steady.dat.'
        close (unit = 7)
600     format(1x,i5,8(3x,f14.7))
650     format(1x,'NACA number:',2x,i5)
651     format(1x,'Number of nodes on lower surface:',2x,i5)
652     format(1x,'Number of nodes on upper surface:',2x,i5)
653     format(1x,'Total number of nodes:',2x,i5)
654     format(1x,'Free stream velocity:','(','units length/units time',
    +           ')',2x,f12.4)
```

```
655     format(1x,'Angle of Attack:','(','degrees',
     +           ')',2x,f12.4)
656     format(1x,'Section lift coefficient:',2x,f12.4)
657     format(1x,'Section drag coefficient:',2x,f12.4)
658     format(1x,'Section moment coefficient:',2x,f12.4)
659     format(1x,'Time:','(','seconds',
     +           ')',2x,f12.4)
660     format(1x,'Circulation:',2x,f12.4)
700     format(1x,3x,'Element',5x,'x',12x,'y',12x,'xbar',12x,'ybar',
     +   12x,'phi',12x,'phiold',12x,'q',12x,'cpsteady')
        return
        end
        subroutine reader(nu,nl,nn,title, x, y,IGF)
        dimension x(nn+1), y(nn+1)
        character *40, title
        open (unit = 7, file = 'xy.dat', status = 'old')
        rewind(unit=7)
        read(7,101) title
        read(7,600) nl
        read(7,600) nu
        read(7,600) nn
        read(7,600) IGF
        do 106    i = 1,nn+1
             read (7,100) x(i), y(i)
106     continue
        close (unit = 7)
101     format(A)
600     format(1x, i5)
700     format(1x, f10.4)
100     format(1x,2(3x,f12.7))
        return
        end
        subroutine rk(nn,nj,nt,y,u,v,B,istart,k,B0,acc)
        dimension y(nn+1,nj), u(nn+1,nj,2), v(nn+1,nj,2)
        dimension z(3), w(3,7)
        real nu
        external derivs
        external rkqc
        print *, 'Creating the stagnation point flow.'
        call findstart(eta,fdp,acc)

        do j = 1, nj
        xs = y(istart,1)*sqrt(B/nu(air))
        xend = y(istart,j)*sqrt(B/nu(air)) - xs
        ifail = 0
        tol= 1.*(10.**(-5.))
        z(1) = 0.
        z(2) = 0.
        z(3) = fdp
        h1 = xend /50.
c-----------------------------------------------------------------
c       Note at this point, the routine has returned the values for
c       F, F', and F'' at eta = y(1,j)*sqrt(B/nu(air)) in the array z
c-----------------------------------------------------------------
        call ODEINT(z,3,xs,xend,tol,H1,HMIN,NOK,NBAD,DERIVS,RK
```

```
     *QC,acc)
        v(istart, j, 2) = -z(1) * sqrt(B*nu(air))
        if (y(istart,j)*sqrt(B/nu(air)) .ge. eta) then
        do l=j+1,nj
        v(istart,l,2) = v(istart,l-1,2) -
     +        (B*(y(istart,l)-y(istart,l-1)))
        end do
        go to 25
        endif
        end do
25      return
        end

        subroutine findmax(tau,u,v,dmax,itau,iu,iv,nn,nj,nt,
     +        istat,iturb,istart,k,isign,imax)
        dimension u(nn+1,nj,2),v(nn+1,nj,2)
        dimension tau(nn+1,2)
        imax = 0
        dmax = 0.0
        dtau = 0.0
        du = 0.0
        dv = 0.0
        do m = istart+isign, istat, isign
        if (abs(tau(m,2)-tau(m,1)) .gt. dtau) then
             dtau = abs(tau(m,2)-tau(m,1))
             itau = m
        endif
        tau(m,1) = tau(m,2)
        do n = 1, nj
        if (abs(u(m,n,2)-u(m,n,1)) .gt. du) then
        du = abs(u(m,n,2)-u(m,n,1))
        iu = m
        endif
        if (abs(v(m,n,2)-v(m,n,1)) .gt. dv) then
        dv = abs(v(m,n,2)-v(m,n,1))
        iv = m
        endif
        u(m,n,1) = u(m,n,2)
        v(m,n,1) = v(m,n,2)
        end do
        end do
        dmax = 0.0
        if (dtau .ge. dmax) then
        dmax = dtau
        imax = itau
        endif
        if (du .ge. dmax) then
        dumax = du
        imax = iu
        endif
        if (dv .ge. dmax) then
        dmax = dv
        imax =iv
        endif
        return
```

```
      end



      subroutine rk2(nn,nj,nt,y,u,v,B,istart,k,B0,acc,isign,x,vtan)
      dimension y(nn+1,nj), u(nn+1,nj,2), v(nn+1,nj,2)
      dimension x(nn+1,nj),vtan(nn+1,2)
      dimension z(3), w(3,7)
      real nu
      external derivs
      external rkqc
      print *, 'Creating the stagnation point flow.'
      call findstart(eta,fdp,acc)

      do j = 1, nj
      xs = y(istart+isign,1)*sqrt(B/nu(air))
      xend = y(istart+isign,j)*sqrt(B/nu(air)) - xs
      ifail = 0
      tol= 1.*(10.**(-5.))
      z(1) = 0.
      z(2) = 0.
      z(3) = fdp
      h1 = xend /50.
c-----------------------------------------------------------------
c     Note at this point, the routine has returned the values for
c     F, F', and F'' at eta = y(1,j)*sqrt(B/nu(air)) in the array z
c-----------------------------------------------------------------
      call ODEINT(z,3,xs,xend,tol,H1,HMIN,NOK,NBAD,DERIVS,RK
     *QC,acc)
      v(istart+isign, j, 2) = -z(1) * sqrt(B*nu(air))
      u(istart+isign, j, 2) = z(2) * B*x(istart+isign,1)
      if (y(istart+isign,j)*sqrt(B/nu(air)) .ge. eta) then
      do l=j+1,nj
      v(istart+isign,l,2) = v(istart,l-1,2) -
     +       (B*(y(istart,l)-y(istart,l-1)))
      u(istart+isign,l,2) = vtan(istart+isign,2)
      end do
      go to 25
      endif
      end do
 25   return
      end

c-----------------------------------------------------------------
c     these are functions the grid generation program will need
c-----------------------------------------------------------------
      subroutine setup(nn,nj,istag,rlen,vtanp,x,vtanbl,k,nt,y,ratio,
     +  h,yftop,yfbottom,ifluid,xoldg,yold,dxdt,dydt,deltat,
     +  ksteady)
      dimension x(nn+1,nj), rlen(nn),
vtanp(nn),vtanbl(nn+1,2),y(nn+1,nj)
      dimension h(nn+1,2),xoldg(nn+1,nj),yold(nn+1,nj),dxdt(nn+1,nj)
      dimension dydt(nn+1,nj)
      real len1,len2
```

```
c-----------------------------------------------------------------
c       here we make the x coordinates
c
c       we have to figure out the origin
c-----------------------------------------------------------------
        totallen = (rlen(istag) + rlen(istag-1))/2.
        fract1 = abs(vtanp(istag-1)/(vtanp(istag)-vtanp(istag-1)))
        fract2 = abs(vtanp(istag)/(vtanp(istag)-vtanp(istag-1)))
        len1 = fract1 * totallen
        len2 = fract2 * totallen
        do i = 1,nn+1
        do j = 1,nj
        xoldg(i,j) = x(i,j)
        yold(i,j) = y(i,j)
        enddo
        enddo
c-----------------------------------------------------------------
c       do the top coordinates
c-----------------------------------------------------------------

        x(istag,1) = 0.0

        x(istag+1,1) = len2
        do 30 i = istag+1, nn
        x(i+1,1) = x(i,1) + (rlen(i-1)/2.) +(rlen(i)/2.)
 30     continue


        totallen = 0.0
        do 45 i=1,nn
        totallen = totallen + rlen(i)
 45     continue
        totallen = totallen-rlen(nn)/2. -rlen(1)/2.

        toplen= 0.0
        do 65 i = istag+1, nn+1
        toplen = toplen+ x(i,1)-x(i-1,1)
 65     continue
c-----------------------------------------------------------------
c       and do the bottom coordinates
c-----------------------------------------------------------------
        x(istag-1,1) = -len1
        do 20 i = istag-2, 1, -1
        x(i,1) = x(i+1,1) - rlen(i)/2.-rlen(i+1)/2.
 20     continue

c-----------------------------------------------------------------
c       next, we put the surface in at y(i,1)
c-----------------------------------------------------------------
        do 70 i   = ifluid, nn+1
        y(i,1) = h(i,2)
 70     continue

c-----------------------------------------------------------------
c       for each x station, we need to find the distance between h(x) and
```

```
c       yf to set the grid
c-----------------------------------------------------------------
        do 60 i   = istag, nn+1
             sum = 0.0
             do 100 j = 0, nj-2
             sum = sum + ratio**(j)
 100         continue
c            dy = (yftop-y(i,1))/sum
             dy = (yftop)/sum

        do 50 j   = 2, nj
        y(i,j) = y(i,j-1) + ratio**(j-2) * dy
 50     continue
 60     continue

        do 160 i = 1, istag-1
             sum = 0.0
             do 101 j = 0, nj-2
             sum = sum + ratio**(j)
 101         continue
c            dy = (yfbottom-y(i,1))/sum
             dy = (yfbottom)/sum
        do 150 j = 2, nj
        y(i,j) = y(i,j-1) + ratio**(j-2) * dy
 150    continue
 160    continue
c-----------------------------------------------------------------
c       now fill in all the x's
c-----------------------------------------------------------------
        do 210 i = 1, nn+1
        do 200 j = 1, nj
        x(i,j) = x(i,1)
 200    continue
 210    continue

        do i = 1,nn+1
        do j = 1,nj
        dxdt(i,j) = (x(i,j)-xoldg(i,j))/deltat
        dydt(i,j) = (y(i,j)-yold(i,j))/deltat
        if (i .eq. istag-1 .or. i .eq. istag .or. i .eq. istag+1) then
        dxdt(i,j) = 0.
        dydt(i,j) = 0.
        endif
        enddo
        enddo

        if (k-ksteady .eq. 0) then
        do i = 1,nn+1
        do j = 1,nj
        xoldg(i,j) = x(i,j)
        yold(i,j) = y(i,j)
        dxdt(i,j) = 0.
        dydt(i,j) = 0.
        enddo
        enddo
```

```
        endif
        return
        end
c-----------------------------------------------------------------
c       subroutine soln
c
c       this routine determines the geometry and the
c       coefficients of the a matrix and the b matrix
c-----------------------------------------------------------------
        subroutine soln(nn, vinf, attack, b, theta, a, bigwork, ipvt,
     +       work, q,rlen,k,circ,circx,circy,nt,xbar,ybar,
     +       deltat,vnorm,vrotate,kgood)
        dimension b(nn+1), theta(nn), a(nn+1,nn+1), bigwork(nn+1,nn+1)
        dimension ipvt(nn+1), work(nn+1), q(nn+1),rlen(nn),circ(nt)
        dimension circx(nt,nt),circy(nt,nt),xbar(nn),ybar(nn),vnorm(nn)
        pi = atan(1.)*4.
        kutta = nn +1

        do 200 i = 1, nn+1
        do 300 j = 1, nn+1
        bigwork(i,j) = a(i,j)
  300   continue
  200   continue


c-----------------------------------------------------------------
c       Calulate the b vector
c-----------------------------------------------------------------
        alpha = attack * (pi/180.)
        do 160 i = 1, nn
             b(i) = vinf * sin (theta(i) - alpha) + vnorm(i)
 160    continue
c-----------------------------------------------------------------
c       this is the part that calculates the additional correction
c       for the normal component of velocity at each node.  it will
c       be used to modify the rhs of the solution vector
c-----------------------------------------------------------------
        if (k .gt. 1) then
        do 30 m = 1, k-1
        do 40 i = 1, nn
        dx = circx(m,k) - xbar(i)
        dy = circy(m,k) - ybar(i)
        f = sqrt(dx*dx + dy*dy)
        vgam = circ(m) / ((2.*pi) * f)
        thetahat = atan2(ybar(i)-circy(m,k),xbar(i)-circx(m,k))
        vn = vgam * cos(thetahat - theta(i))
        b(i) = b(i) + vn
 40     continue
 30     continue
        endif
c-----------------------------------------------------------------
c       Calculate the tangency condition at the trailing edge
c-----------------------------------------------------------------
        rone = cos (theta(1)- alpha )
        two = cos (theta(nn) - alpha)
```

```
        b (kutta) = - vinf * (rone + two)
c-----------------------------------------------------------------
c       this part calculates the correction to the b(nn+1) term to
c       be used in the modified solution.
c-----------------------------------------------------------------
        if (k .gt. 1) then
        do 20 m = 1, k-1
        dx1 = xbar(1) - circx(m,k)
        dxn = xbar(nn) - circx(m,k)
        dy1 = ybar(1) - circy(m,k)
        dyn = ybar(nn) - circy(m,k)
        r1 = sqrt(dx1*dx1 + dy1*dy1)
        rn = sqrt(dxn*dxn + dyn*dyn)
        vgam1 = circ(m) / ((2.*pi) * r1)
        vgamn = circ(m) / ((2.*pi) * rn)
        thetahatn = atan2(dyn,dxn)
        thetahat1 = atan2(dy1,dx1)
        rhs = -vgam1 * (sin(thetahat1-theta(1))) -vgamn * (sin(
     +       thetahatn-theta(nn)))
        b(kutta) = b(kutta) + rhs
 20     continue
        endif
c-----------------------------------------------------------------
c       and solve the system of equations
c-----------------------------------------------------------------

        condt = 0
        call decomp(nn+1,nn+1,bigwork,condt,ipvt,work)
c       print *, 'condition number = ', condt
        call solve(nn+1,nn+1,bigwork,b,ipvt)
c-----------------------------------------------------------------
c       these are checks...
c-----------------------------------------------------------------
        sources = 0.0
        do 400 i = 1, nn
             sources = sources + (b(i) * rlen(i))
 400    continue
c       print *, 'Sum of sources:',sources
        do 405 i = 1, kutta
             q(i) = b(i)
 405    continue
c-----------------------------------------------------------------
c       this part calculates the circulation at each time step
c       that is a result of the solution reached after the
c       rhs of the matrix equation has been modified.  it also
c       determines the position of the trailing vortex at the next time
c       step.
c-----------------------------------------------------------------
        temp = 0.0
        do 15 i = 1, nn
             temp = temp + rlen(i)*q(nn+1)
 15     continue
c       print *, 'Gamma: ', temp
        if (k .eq. 1) then
             circ(1) = - temp
```

```
          go to 999
        endif

        totalcirc = 0.0
        do 25 m = 1, k-1
             totalcirc = totalcirc +circ(m)
25      continue
        circ(k) = - temp - totalcirc
        temp2 = circ(k)
        if (kgood .ne. 0) then
        circ(k) = 0.
        circ(kgood) = (circ(kgood) + temp2)/2.
        endif
999     if (abs(circ(k))/vinf .le. 1E-06) then
             circ(k) = 0.0
        endif
        return
        end


        subroutine start3(nn,vinf,vnorm,xbar,istag,dvnorm,vnormold,
     +  vnormold2,sor)
        dimension vnorm(nn),xbar(nn),dvnorm(nn),vnormold(nn)
        dimension vnormold2(nn)
        do i = 1,nn
             vnorm (i) = (vnormold2(i) + vnormold(i))/2.
        enddo
        return
        end



        subroutine start(nn,vinf,vnorm,xbar,inorm)
        dimension vnorm(nn),xbar(nn)
        do i = 1, nn
        vnorm(i) = 0.05*vinf
        enddo
        if (inorm .eq. 1) then
        open (unit = 16, file = 'normstart.dat', status = 'unknown')
        do i = 1, nn
        read (16,25) i, vnorm(i)
        enddo
        close(16)
        endif
25      format(1x, i5,3x,f12.7)
        return
        end

        subroutine normwriter(nn,vnorm)
        dimension vnorm(nn)
        open (unit = 16, file = 'normstart.dat', status = 'unknown')
        do i = 1, nn
        write (16,25) i, vnorm(i)
        enddo
        close(16)
```

```
 25     format(1x, i5,3x,f12.7)
        return
        end

        subroutine start2(nn,vinf,vnorm,xbar,dvnorm,attack,sor)
        dimension vnorm(nn),xbar(nn),dvnorm(nn)
        do i = 1, nn
             vnorm(i) = (vnorm(i) - (dvnorm(i))) + dvnorm(i)*sor
        enddo
        return
        end


        subroutine changev(vinf,v0,vnorm,nn)
        dimension vnorm(nn)
        do i = 1, nn
        vnorm(i) = vnorm(i) * vinf/(v0)
        enddo
        return
        end
c-----------------------------------------------------------------
c       subroutine solvit
c
c       this subroutine solves the boundary layer and depth of deicing
c       fluid at a position at a given time.
c-----------------------------------------------------------------
        subroutine solvit(k,nn,nj,nt,u,v,vtan,bb,dd,aa,cc,y,
     +       deltat,x,tau, delta, delta2, delta3, shape,
     +       yftop, yfbottom, iyfflag,li,lo,lm,yplus,h,ksteady,
     +       ratio,yold,uold,vold,isign,vnorm,dvnorm,xwork,
     +       vnormwork,normalflag,isteady,istag,      !this part is for
usolve
     +       dtau,dus,                        !these two are misc.
     +       f,aaf,bbf,ddf,                          !these are for dfluid
     +       work1,work2,work3,ifluid,istopf,muratio,tauw,vinf,t,
     +       markthrough,ineg,icrap,vnormold,vnormold2,xoldg,yoldg,
     +       dxdt,dydt,iseplamt,iseplamb,itransflagt,itransflagbegt,
     +       itransflagb,itransflagbegb,xbar,sor,inorm,htemp,vrotate,
     +       attack,ho,vnormok,ifinal,isept,isepb,itsover)
c-----------------------------------------------------------------
c       these first ones are for usolve
c-----------------------------------------------------------------
        dimension u(nn+1,nj,2), v(nn+1,nj,2), vtan(nn+1,2),xbar(nn)
        dimension aa(nj), bb(nj), cc(nj), dd(nj),vnormold2(nn)
        dimension x((nn+1),nj),y((nn+1),nj), tau((nn+1),2),
delta((nn+1),2)
        dimension delta2((nn+1),2), delta3((nn+1),2), shape((nn+1),2)
        dimension yold(nn+1,nj), vold(nn+1,nj), uold(nn+1,nj)
        dimension
vnorm(nn),dvnorm(nn),xwork(nn),vnormwork(nn),vnormold(nn)
        dimension xoldg(nn+1,nj), yoldg(nn+1,nj),htemp(nn+1,2)
        dimension vnormok(nn)
        real li(nj), lo(nj), lm(nj), yplus(nj), h((nn+1),2), tauw(nn+1,2)
        real nu, pplus, muratio
c-----------------------------------------------------------------
```

```
c       these are miscellaneous
c-----------------------------------------------------------------
        dimension dus(nn+1), dtau(nn+1)
        dimension dxdt(nn+1,nj),dydt(nn+1,nj)
c-----------------------------------------------------------------
c       this is for dfluid
c-----------------------------------------------------------------
        dimension f(nn+1), aaf(nn+1)
        dimension ddf(nn+1), work1(4), work2(4), work3(3)
        dimension bbf(nn+1)
        print *, 'Relaxation factor: ', sor
        iterice = 0
        itericemax = 25
        eps = (attack)*2.*(10.**(-3.))
        eps2 = 1.*(10.**(-7.))
        normalflag = 0
        isept = 0
        isepb = 0
        iseplamt = 0
        iseplamb = 0
        ineg = 0
        icrap = 0
        do 5 i = 1, nn+1
        u(i,1,2) = u(i,1,1)
        tau(i,2) = tau(i,1)
 5      enddo
        if (markthrough .eq. 0) then
             do 6 i = ifluid, nn+1
             h(i,2) = h(i,1)
 6           enddo
             do i = 1, nn+1
             tau(i,2) = tau(i,1)
             u(i,1,2) = u(i,1,1)
             enddo
             markthrough = 1
        endif

 75     do i = 1, nn+1
        dtau(i) = tau(i,2)
        dus(i) = u(i,1,2)
        enddo
        call usolve(k,nn,nj,nt,u,v,vtan,bb,dd,aa,cc,y,
     +       deltat,x, tau, delta, delta2, delta3, shape,
     +       yftop,iyfflag,li,lo,lm,yplus,h,istag,nn+1,ksteady,
     +       iturb,ratio,yold,uold,vold,1,vnorm,dvnorm,xwork,
     +       vnormwork,normalflag,isteady,istag,isept,t,
     +       dxdt,dydt,xoldg,yoldg,iseplamt,itransflagt,itransflagbegt,
     +       xbar)

c       if (iseplamt .ne. 0) return
        print *, 'Top boundary layer solved.'
        if (k-ksteady .ne. 0 .and. ho .ne. 0.) then
             iterice = iterice +1
             call dfluid(k,nn,nt,h,tau,vtan,f,aaf,bbf,ddf,
     +       work1,work2, work3,x,deltat,u,nj,ifluid,istopf,
```

```
     +        muratio,ksteady,t,icrap,ineg,vinf,dxdt,sor,htemp,
     +        vrotate)

              print *, 'Deicing fluid module solved.'

              if(iterice .gt. itericemax) then
                   print *, 'Too many tries...'
                   iseplamb = 1
                   return
              endif

35            call initialgrid(nn,nj,ratio,yftop,x,y,h,nt,k,istag,
     +             ifluid,yoldg,dydt,deltat)
          endif

          do 20 i = 1, nn+1
          dtau(i) = dtau(i) - tau(i,2)
20        continue
          do 30 i = 1,nn+1
          dus(i) = dus(i) - u(i,1,2)
30        continue

          difftaumax = 0.0
          diffusmax = 0.0
          do 40 i = 1,nn+1
          if (abs(dtau(i)) .gt. difftaumax) then
               difftaumax = abs(dtau(i))
          endif
          if (abs(dus(i)) .gt. diffusmax) then
               diffusmax = abs(dus(i))
          endif
40        continue
          diffmax = max(difftaumax,diffusmax)
          if (isept .ne. 0 .and. diffmax .ge. eps ) then
          print *, 'Separated but not converged...'
          isept = 0
          go to 75
          endif
          if (isept .ne. 0 .and. diffmax .le. eps) then
          go to 90
          endif

          if(diffmax .ge. eps ) then
          print *, 'Diffmax in deicing-gas iteration:',diffmax
          endif

          if (diffmax .lt. eps .and. isept .eq. 0) then
               print *, 'New grid being formed.'
               call initialgrid(nn,nj,ratio,yftop,x,y,h,nt,k,istag,
     +              ifluid,yoldg,dydt,deltat)
               go to 90
          endif
          go to 75
90        print *, 'Working on bottom boundary layer.'
          if(abs(h(ifluid+1,2)) .le. eps2) then
```

```
            ifluid = ifluid +1
          endif
          call usolve(k,nn,nj,nt,u,v,vtan,bb,dd,aa,cc,y,
     +         deltat,x, tau, delta, delta2, delta3, shape,
     +         yfbottom, iyfflag,li,lo,lm,yplus,h,istag,1,ksteady,
     +         0,ratio,yold,uold,vold,-1,vnorm,dvnorm,xwork,
     +         vnormwork,0,isteady,istag,isepb,t,
     +         dxdt,dydt,xoldg,yoldg,iseplamb,itransflagb,itransflagbegb,
     +         xbar)


          print *, 'Bottom boundary layer solved.'
          if (iseplamb .ne. 0) return
          if (k-ksteady .eq. 0 .and. inorm .eq. 1) then
          go to 47
          endif
          iseplamb = 0
          call vnormal(vnorm, vtan, delta2, x, nn, nj, nt,
     +         normalflag,k,deltat,dvnorm,xwork,vnormwork,istag,h,
     +         vinf,t,vnormold,vnormold2,isept,isepb,sor,ksteady,
     +         iseplamb,attack)
 47       if (iseplamb .eq. 1) return
          icrap = 0

          if ((isept .ne. 0 .or. isepb .ne. 0) .and.
     +         normalflag .eq. 0) then
               print *, 'It really separated.'
               call output1(k+1, nn+1, nj, nt, deltat, x, tau, delta,
     +         delta2, delta3, shape,vtan,h,istag,nn+1,1,u,t)
          call output2(k+1, nn+1, nj, nt, deltat, x, y, delta,u,v,
     +         vtan,tau,istag,nn+1,1,t)
               call output1(k+1, nn+1, nj, nt, deltat, x, tau, delta,
     +         delta2, delta3, shape,vtan,h,istag,1,-1,u,t)
          call output2(k+1, nn+1, nj, nt, deltat, x, y, delta,u,v,
     +         vtan,tau,istag,1,-1,t)
               itsover = 1
               return
          endif
c         if (abs(itransflagbegb - itransflagb) .gt. 2 .and.
c     +           k-ksteady .ne. 0) then
c              iseplamb = 1
c              itransflag = itransflagbeg
c              call fixit(nn,vnorm,vnormold2,vnormold)
c              normalflag = 1
c              print *, 'Final transition differs by more than'
c              print *, 'two from the previous time step on the bottom...'
c         endif
c         if (abs(itransflagbegt - itransflagt) .gt. 3 .and.
c      +           k-ksteady .ne. 0) then
c              iseplamt = 1
c              itransflagt = itransflagbegt
c              call fixit(nn,vnorm,vnormold2,vnormold)
c              normalflag = 1
c              print *, 'Final transition differs by more than'
c              print *, 'two from the previous time step on the top...'
```

```
c        endif
         return
         end

c-----------------------------------------------------------------

         subroutine fixit(nn,vnorm,vnormold2,vnormold)
         dimension vnorm(nn), vnormold2(nn),vnormold(nn)
         do i = 1, nn
         vnorm(i) = (vnormold2(i) + vnormold(i))/2.
         enddo
         return
         end
c-----------------------------------------------------------------
c        subroutine tanvel
c
c        calculates the tangential velocities, pressures and pressure
coeff.
c-----------------------------------------------------------------
         subroutine tanvel(nn,vtan,vinf, theta, attack, r, q, phi,phiold,
     +        deltat,cp,beta,k,circ,circx,circy,nt,x,y,xbar,ybar,eta)
         dimension vtan(nn), theta(nn), r(nn, nn+1), q(nn+1), phi(nn)
         dimension phiold(nn),cp(nn),beta(nn,nn),circ(nt),circx(nt,nt)
         dimension circy(nt,nt),x(nn+1),y(nn+1),xbar(nn),ybar(nn)
         pi = atan(1.)*4.
         alpha = attack*(pi/180.)
c-----------------------------------------------------------------
c        Now, calculate the tangential velocities
c-----------------------------------------------------------------
         do 410 i = 1,nn
              vtan(i) = vinf * cos(theta(i) - alpha)

              vsource = 0.0
              do 420 j = 1, nn
                   rone = sin (theta(i) -theta(j))
                   two = cos (theta(i) -theta(j))
                   three = alog(r(i,j+1)/r(i,j))
                   vsource = vsource + ((q(j)/(2*pi)) * (rone*beta(i,j) -
     +                   (two*three)))
 420          continue
              vtan(i) = vtan(i) + vsource
              vcirc = 0.0
              do 430 j = 1, nn
                   rone = sin (theta(i) -theta(j))
                   two = cos (theta(i) -theta(j))
                   three = alog(r(i,j+1)/r(i,j))
                   four = q(nn+1)/(2*pi)
                   five = (rone*three + two*beta(i,j))
                   vcirc = vcirc + (five * four)
 430          continue
         vtan(i) = vtan(i) + vcirc
 410     continue
c-----------------------------------------------------------------
c        this part of the program calculates the additional tangential
c        velocity at each node due to the trailing vortices
```

```
c-----------------------------------------------------------------------
        if (k .gt. 1) then
        do 300 m = 1, k-1
        do 110 i = 1, nn
        dx = xbar(i) - circx(m,k)
        dy = ybar(i) - circy(m,k)
        f = sqrt(dx*dx + dy*dy)
        vgam = circ(m) / ((2.*pi) * f)
        thetahat = atan2(dy,dx)
        vtan(i) = vtan(i) + vgam*sin(thetahat-theta(i))
 110    continue
 300    continue
        endif
c-----------------------------------------------------------------------
c       this part of the calculation determines the velocity at a
c       point due to all the sources, the distributed circulation,
c       the freestream, and the trailing vortices
c-----------------------------------------------------------------------
        if (k .gt. 1) then
        do 80 m = 1, k-1
        u = vinf*cos(alpha)
        v = vinf*sin(alpha)
        usv = 0.0
        vsv = 0.0
        do 70 j = 1, nn
             rone = circy(m,k) - y(j+1)
             two = circx(m,k) - x(j)
             three = circx(m,k) - x(j+1)
             four = circy(m,k) - y(j)
             rnum = (rone * two) - (three * four)
             den = (three * two) + (rone * four)
             zeta = atan2(rnum,den)

             dy = y(j) - circy(m,k)
             dx = x(j) - circx(m,k)
             r2 = sqrt((dy*dy) + (dx*dx))
             dy = y(j+1) - circy(m,k)
             dx = x(j+1) - circx(m,k)
             r1 = sqrt((dy*dy) + (dx*dx))

             usstar = -(1./(2.*pi)) * alog(r1/r2) * q(j)
             vsstar = (1./(2.*pi)) * zeta * q(j)
             uvstar = (1./(2.*pi)) * zeta * q(nn+1)
             vvstar = (1./(2.*pi)) * alog(r1/r2) * q(nn+1)

             ustar = usstar + uvstar
             vstar = vsstar + vvstar

             usv = usv + ustar*cos(theta(j)) - vstar*sin(theta(j))
             vsv = usv + ustar*sin(theta(j)) + vstar*cos(theta(j))

 70     continue
             u = u + usv
               v = v + vsv
```

```
c-----------------------------------------------------------------
c       here, the contributions from all sources and distributed vortices
c       are included in the velocity at position circx(m,k),circy(m,k).
c       the contribution from the freestream has also been included
c
c       m is at while n is from
c
c       Now, we need the contribution from each of the vortices
c-----------------------------------------------------------------
        do 100 n = 1, k-1
        if (m .ne. n) then
        dx = circx(m,k) - circx(n,k)
        dy = circy(m,k) - circy(n,k)
        f = sqrt(dx*dx + dy*dy)
        vgam = circ(n)/(2.*pi*f)

        thetahat = atan2(dy,dx)
        u = u + vgam*sin(thetahat)
        v = v - vgam*cos(thetahat)
        endif
 100    continue
        circx(m,k+1) = circx(m,k) + u*deltat
        circy(m,k+1) = circy(m,k) + v*deltat
 80     continue
        endif

        do 440 i = 1, nn
        rone = (vtan(i)/vinf)**2.
        rtwo = 2.*(((phi(i) - phiold(i))/deltat)/(vinf**2.))
        cp(i) = 1 - rone - rtwo
 440    continue
        circx(k,k+1) =  1. + (vinf*deltat)*cos(eta)
        circy(k,k+1) =   (vinf*deltat)*sin(eta)
c       print *,'Tangential velocities and pressure coefficients
calculated.'
        return
        end
c-----------------------------------------------------------------
c       subroutine sy
c-----------------------------------------------------------------
        subroutine sy(il,iu,bb,dd,aa,cc,nn)
        dimension aa(nn), bb(nn), cc(nn), dd(nn)
c-----------------------------------------------------------------
c       subrooutine sy solves tridiagonal system by elimination
c       il = subscript of first equation
c       iu = subscript of last equation
c       bb = coefficient behind diagonal
c       dd = coefficient on diagonal
c       aa = coefficient ahead of diagonal
c       cc = element of constant vector
c
c       establish upper triangular matirix
c-----------------------------------------------------------------
        lp = il+1
        do 1000 i = lp,iu
```

```
        r = bb(i)/dd(i-1)
        dd(i) = dd(i) - r*aa(i-1)
 1000   cc(i) = cc(i) - r*cc(i-1)
c-----------------------------------------------------------------
c       back substitution
c-----------------------------------------------------------------
        cc(iu) = cc(iu)/dd(iu)
        do 2000 i = lp,iu
        j = iu - i + il
 2000   cc(j) = (cc(j) -aa(j) *cc(j+1))/dd(j)
c-----------------------------------------------------------------
c       solution stored in cc
c-----------------------------------------------------------------
        return
        end  subroutine transfervtan(vtanp, vtanbl, k, istag, nn, nt)
        dimension vtanp(nn), vtanbl(nn+1,2)

        do 10 i = 1, istag-1
        vtanbl(i,2) = vtanp(i)
10      continue

        vtanbl(istag,2) = 0.0

        do 20 i = istag+1, nn+1
        vtanbl(i,2) = vtanp(i-1)
20      continue

        return
        end
c-----------------------------------------------------------------
c       this subroutine solves for the velocities at time step 2
c       it assumes the boundary layer equations are applicable
c
c-----------------------------------------------------------------
        subroutine usolve(k,nn,nj,nt,u,v,vtan,bb,dd,aa,cc,y,
     +       deltat,x, tau, delta, delta2, delta3, shape,
     +       yf, iyfflag,li,lo,lm,yplus,h,istart,istop,ksteady,
     +       iturb,ratio,yold,uold,vold,isign,vnorm,dvnorm,xwork,
     +       vnormwork,normalflag,isteady,istag,isep,t,dxdt,dydt,
     +       xoldg,yoldg,iseplam,itransflag,itransflagbeg,xbar)
        dimension u(nn+1,nj,2), v(nn+1,nj,2), vtan(nn+1,2),xbar(nn)
        dimension aa(nj), bb(nj), cc(nj), dd(nj)
        dimension x((nn+1),nj),y((nn+1),nj), tau((nn+1),2),
delta((nn+1),2)
        dimension delta2((nn+1),2), delta3((nn+1),2), shape((nn+1),2)
        dimension yold(nn+1,nj), vold(nn+1,nj), uold(nn+1,nj)
        dimension vnorm(nn),dvnorm(nn),xwork(nn), vnormwork(nn)
        dimension dxdt((nn+1),nj),dydt((nn+1),nj),xoldg((nn+1),nj)
        dimension yoldg((nn+1),nj)
        real li(nj), lo(nj), lm(nj), yplus(nj), h((nn+1),2)
        real nu, pplus
        imax = 0
        isep = 0
        iseplam = 0
        if (isign .eq. 1) then
```

```
        istat = 0
        endif
        if (isign .eq. -1) then
        istat = 999
        endif
        iturb = 0
c-----------------------------------------------------------------
c       at any particular time, the u velocity at the stagnation point is
c       zero.  we approximate the stagnation point flow by adding the
values
c       for v at this point.
c-----------------------------------------------------------------
 17     B = (vtan(istart+1,2)-vtan(istart,2))/
     +        (x(istart+1,1)-x(istart,1))
        if (k-ksteady .eq. 0) then
        do i = istart,istop, isign
        vtan(i,1) = vtan(i,2)
        end do
        endif

        B0 = (vtan(istart+1,1)-vtan(istart,1))/
     +        (x(istart+1,1)-x(istart,1))
c       acc = (B - B0)/(deltat * (B**2.))
        acc = 0.
c-----------------------------------------------------------------
c       this gets the proper v velocity at the origin of the stagnation
point
c-----------------------------------------------------------------
c       if (isign .eq. 1) then
c       call changegrid(v,y,nn,nj,nt,istart,k,li,lo,isign)
c       call rk(nn,nj,nt,y,u,v,B,istart,k,B0,acc)
c       endif
c       if (isign .eq. -1) then
c       call changegrid(v,y,nn,nj,nt,istart,k,li,lo,isign)
c       endif
        call rk2(nn,nj,nt,y,u,v,B,istart,k,B0,acc,isign,x,vtan)
        if (k-ksteady .eq. 0) then
        do j = 1,nj
        u(istart+isign,j,1) = u(istart+isign,j,2)
        v(istart+isign,j,1) = v(istart+isign,j,2)
        enddo
        endif

c-----------------------------------------------------------------
        if (isign .eq. 1) then
        open (unit=27, file = 'staginfotop.dat', status = 'unknown')
        endif
        if (isign .eq. -1) then
        open (unit=27, file = 'staginfobot.dat', status = 'unknown')
        endif
        write (27,*)'Time:', t
        write (27,*)'deltat:',deltat
        write (27,*)'at station:',istag
        write (27,*) 'Region of interest:',yf
        write (27,*)'Tangential velcoity at istag:',vtan(istag,2)
```

```
      write (27,*)'Previous tangential velocity:',vtan(istag,1)
      write (27,246) 'j', 'y','uij1','vij1','uij2','vij2'
      do m = 1,nj
      write(27,245) m, y(istag,m), u(istag,m,1),
     +         v(istag,m,1),
     +         u(istag,m,2),v(istag,m,2)
      enddo
 245  format(1x,i3,3x,5(E12.5,3x))
 246  format(1x,6(3x,A12))
      close(27)
c-----------------------------------------------------------------

c-----------------------------------------------------------------
c       here, we set the outer velocities to vtan
c-----------------------------------------------------------------
      do 5 i = istart+(isign+isign), istop, isign
      u(i,nj,2) = vtan(i,2)
 5    continue

 75   eps = 1.0*10.**(-2.)
      do 10 i = istart+(isign+isign), istop, isign
      mod = 0
c-----------------------------------------------------------------
c       here the marching is in the i direction at any particular time, t
c       we are looking to get a solution for u(i,j,2)
c-----------------------------------------------------------------
      dx = x(i,1) - x(i-isign,1)
      do 20 j = 2, nj-1

      dy = (y(i,j) - y(i,j-1))
      beta = (y(i,j+1) - y(i,j))/(y(i,j) - y(i,j-1))

      bb(j) = (-deltat * (v(i-isign,j,2)+dydt(i,j)) *
     +       (beta))/((beta+1.)*dy)
     +       +((-2. * nu(air) * deltat)/((1.+beta)*(dy**2.)))
      aa(j) = (deltat * (v(i-
isign,j,2)+dydt(i,j)))/((beta)*(beta+1.)*dy)
     +       +((-2. * nu(air) * deltat)/((beta)*(1.+beta)*(dy**2.)))
      dd(j) = 1.0 + ((deltat/dx)*(u(i-isign,j,2)+dxdt(i,j))) +
     +       ((deltat * (v(i-isign,j,2)+dydt(i,j)) * ((beta**2.) - 1.))
     +       /((beta+1.)*dy*beta)) +
     +       (2.*nu(air)*deltat/((dy**2.)*beta))
      cc(j) = u(i,j,1) + vtan(i,2) - vtan(i,1) +
     +       deltat*((vtan(i-isign,2)+dxdt(i,1))/dx) * (vtan(i,2) -
     +       vtan(i-isign,2)) +
     +       (deltat/dx)*(u(i-isign,j,2)*(u(i-isign,j,2)+dxdt(i,j)))
c-----------------------------------------------------------------
c       here we put the additional part in for the turbulence if
c       necessary
c-----------------------------------------------------------------
      if (iturb .ne. 0) then
      ustar = (abs(tau(i-isign, 2)) / rho(air))**(1./2.)
c-----------------------------------------------------------------
c       now, we modify the value of A+ based on the pressure gradient
c       as suggested by White, p.442
```

```
c-----------------------------------------------------------------
        dvdt = (vtan(i,2)-vtan(i,1))/deltat
        dvdx = (vtan(i,2)-vtan(i-isign,2))/dx
        dpdx = -rho(air) * isign*(dvdt + ((vtan(i,2)+dxdt(i,1)) * dvdx))
        pplus = (nu(air) * dpdx) / (rho(air) * (ustar**3.))
        const = 11.8
        if (pplus .lt. -0.08) then
        aplus = 260.
        go to 400
        endif
        aplus = 26. / ((1. + (const * pplus))**(1./2.))
c-----------------------------------------------------------------
c       Now we calculate the mixing lengths at each of the stations
c       First, the mixing lengths predicted by the law of the wall
c       along the inner region
c
c       We base the outer law on the boundary layer thickness at the
c       previous station
c-----------------------------------------------------------------
 400    do 76 n = 1, nj
            yplus(n) = (y(i,n)-y(i,1)) * ustar / nu(air)
        eta = (y(i,n)-y(i,1))/delta(i-isign,2)
            xi = dpdx*delta(i-isign,2)/tau(i-isign,2)
        fract = 1. + xi*eta - ((3.+2.*xi)*(eta**2.))
     +       +((2.+xi)*(eta**3.))
            if (fract .le. 0.) then
                fract = 0.
            endif
            fract = sqrt(fract)
            lm(n) = 0.41*(y(i,n)-y(i,1))*(1.-exp(-yplus(n)/aplus))*fract
            if (yplus(n) .lt. 50. .and. mod .eq. 0 .and.
     +          lm(n) .gt. 0.089*delta(i-isign,2)) then
                mod = 1
            endif
            if (lm(n) .gt. 0.089*delta(i-isign,2) .and. mod .eq. 1 .and.
     +          yplus(n) .gt. 50.) then
                temp = 50.*nu(air)/ustar
                lm(n) = 0.41*temp*(1.-exp(-50./aplus))*fract
                lm(n) = 0.089*delta(i-isign,2)
                go to 76
            endif
            if (lm(n) .gt. 0.089*delta(i-isign,2) .and. mod .eq. 0) then
                lm(n) = 0.089*delta(i-isign,2)
                go to 76
            endif

 76     continue
c-----------------------------------------------------------------
c       Now, we find the modified terms of the K matrix
c-----------------------------------------------------------------

 77     dy = y(i,j)-y(i,j-1)
        beta = (y(i,j+1)-y(i,j))/dy

        smla = lm(j)**2. + lm(j+1)**2.
```

```
        smlb = lm(j)**2. + lm(j-1)**2.

        udiffa = abs(u(i-isign,j+1,2) -u(i-isign,j,2))
        udiffb = abs(u(i-isign,j,2) - u(i-isign,j-1,2))

        dysum = dy*(1.+ beta)

        bb(j) = bb(j) -(((deltat*smlb)/dysum)*(udiffb/dy**2.))
        aa(j) = aa(j) -(((deltat*smla)/dysum)*(udiffa/(beta*dy)**2.))
        dd(j) = dd(j) +(((deltat*smlb)/dysum)*(udiffb/dy**2.))
     +              +(((deltat*smla)/dysum)*(udiffa/(beta*dy)**2.))
        endif
 20     continue

        cc(nj-1) = cc(nj-1) -aa(nj-1)*vtan(i,2)
        cc(2) = cc(2) - bb(2)*u(i,1,2)
c-----------------------------------------------------------------
c       call sy(il,iu,bb,dd,aa,cc)
c
c       Use the Thomas algorithm to solve for u at each i station
c-----------------------------------------------------------------
        call sy(2,nj-1,bb,dd,aa,cc,nj)
c-----------------------------------------------------------------
c       recover the solution
c-----------------------------------------------------------------
        do 30 j = 2, nj-1
        u(i,j,2) = cc(j)
 30     continue
c-----------------------------------------------------------------
c       find the v solution
c-----------------------------------------------------------------
        do 40 j = 2, nj
c-----------------------------------------------------------------
c       conservation of mass
c-----------------------------------------------------------------
        dy = (y(i,j) - y(i,j-1))
        v(i,j,2) = v(i,j-1,2) - (dy/(2.*dx))*(u(i,j,2) -
     +  u(i-isign,j,2) +
     +  u(i,j-1,2) - u(i-isign,j-1,2))
 40     continue
        if (k-ksteady .eq. 0 .and. i .eq. istart+isign+isign) then
        do m = 1,nj
        u(istart+isign+isign,m,1) = u(istart+isign+isign,m,2)
        enddo
        endif
c-----------------------------------------------------------------
c       for this particular value of i, calculate if we have shifted to
c       turbulence.  we need the displacement thickness, the momentum
c       thickness and the Reynolds number based on displacement thickness
c-----------------------------------------------------------------

        delta(i,2) = 0.0
        do 200 j = 1, nj
             if (u(i,j,2)/vtan(i,2) .ge. 0.99) then
             delta(i,2) = y(i,j)-y(i,1)
```

```
c-----------------------------------------------------------------
c       the check for the upper surface
c-----------------------------------------------------------------
              if (k-ksteady .eq. 0 .and. (istat
     +             .lt. i .or. i .eq. istop) .and. isign .eq. 1) then
              istat = i
              call findmax(tau,u,v,dmax,itau,iu,iv,nn,nj,nt,
     +        istat,iturb,istart,k,isign,imax)
              if (i .eq. istop) eps = eps/100.
              if (dmax .gt. eps) then
                  if (i .eq. istop) then
                  print *, 'Helping the boundary layer start.'
                  print *, 'At station:', imax
                  print *, 'dmax = ',dmax
                  endif
                  iturb = 0
                  iseplam = 0
                  isep = 0
                  go to 75
              endif
              endif
c-----------------------------------------------------------------
c       the check for the lower surface
c-----------------------------------------------------------------
              if (k-ksteady .eq. 0 .and. (istat
     +             .gt. i .or. i .eq. istop) .and. isign .eq. -1) then
              istat = i
              call findmax(tau,u,v,dmax,itau,iu,iv,nn,nj,nt,
     +        istat,iturb,istart,k,isign,imax)
              if (i .eq. istop) eps = eps/100.
              if (dmax .gt. eps) then
                  if (i .eq. istop) then
                  print *, 'Helping the boundary layer start.'
                  print *, 'At station:', imax
                  print *, 'dmax = ',dmax
                  endif
                  iturb = 0
                  iseplam = 0
                  isep = 0
                  go to 75
              endif
              endif

              if (yf .le. 1.2*(delta(i,2)+y(i,1))) then
              print *, 'Region too small. Trying yf to be ', 1.25*yf
              print *, 'At station',i
              print *, ' '
              call adjust(nn,nj,nt,k,delta,yf,ratio,u,v,
     +             istart,istop,y,
     +        yold,uold,vold,iflag,ksteady,isign,yoldg,dydt,deltat)
              if (isign .eq. 1) then
                  call rk2(nn,nj,nt,y,u,v,B,istart,k,B0,acc,isign,x,vtan)
              endif
c             if (isign .eq. -1) then
c                 call changegrid(v,y,nn,nj,nt,istart,k,li,lo,isign)
```

```
c            endif
             iturb = 0
             iseplam = 0
             iflag = 0
             isep = 0
             go to 75
             endif
             go to 300
             endif
 200     continue

 300     beta = (y(i,3) - y(i,2))/(y(i,2) - y(i,1))
         dy = (y(i,2) - y(i,1))
         rone =    -((3. + 4.*beta + 3.*beta**2. + beta**3.)*u(i,1,2)/
     +   ((1. + 2.*beta + 2.*beta**2. + beta**3.)*dy)) +
     +   (1. + beta + beta**2.)*u(i,2,2)/(beta**2.*dy) -
     +   (1. + beta + beta**2.)*u(i,3,2)/(beta**3.*dy + beta**4.*dy) +
     +   u(i,4,2)/(beta**3.*(1. + beta + beta**2.)*dy)
         tau(i,2) = visc(air) * rone * isign
         l = istart+isign
         beta = (y(l,3) - y(l,2))/(y(l,2) - y(l,1))
         dy = (y(l,2) - y(l,1))
         rone =  -((3. + 4.*beta + 3.*beta**2. + beta**3.)*u(l,1,2)/
     +   ((1. + 2.*beta + 2.*beta**2. + beta**3.)*dy)) +
     +   (1. + beta + beta**2.)*u(l,2,2)/(beta**2.*dy) -
     +   (1. + beta + beta**2.)*u(l,3,2)/(beta**3.*dy + beta**4.*dy) +
     +   u(l,4,2)/(beta**3.*(1. + beta + beta**2.)*dy)
         tau(l,2) = visc(air) * rone * isign

         delta2(i,2) = 0.0
         do 500 j = 2, nj
              dy = (y(i,j+1) - y(i,j))
              fb = 1.- (u(i,j,2)/vtan(i,2))
              fa = 1.- (u(i,j-1,2)/vtan(i,2))
              delta2(i,2) = delta2(i,2) + (dy * ((fb+fa)/2.))
 500     continue
         delta3(i,2) = 0.0
         do 700 j = 2, nj
              dy = (y(i,j+1) - y(i,j))
              fb = (1.- (u(i,j,2)/vtan(i,2)))*(u(i,j,2)/vtan(i,2))
              fa = (1.- (u(i,j-1,2)/vtan(i,2)))*(u(i,j-1,2)/
     +             vtan(i,2))
              delta3(i,2) = delta3(i,2) + (dy * ((fb+fa)/2.))
 700     continue

         shape(i,2) = delta2(i,2)/delta3(i,2)
         delta(istart+isign,2) = 0.0
         do j = 1, nj
              if (u(istart+isign,j,2)/vtan(istart+isign,2) .ge. 0.99) then
              delta(istart+isign,2) = y(istart+isign,j)-y(istart+isign,1)
              go to 41
              endif
         enddo

 41      delta2(istart+isign,2) = 0.0
```

```
      do 501 j = 2, nj
           dy = (y(istart+isign,j+1) - y(istart+isign,j))
           fb = 1.- (u(istart+isign,j,2)/vtan(istart+isign,2))
           fa = 1.- (u(istart+isign,j-1,2)/vtan(istart+isign,2))
           delta2(istart+isign,2) = delta2(istart+isign,2) +
     +   (dy * ((fb+fa)/2.))
 501     continue
         delta3(istart+isign,2) = 0.0
         do 701 j = 2, nj
           dy = (y(istart+isign,j+1) - y(istart+isign,j))
           fb = (1.- (u(istart+isign,j,2)/vtan(istart+isign,2)))*
     +     (u(istart+isign,j,2)/vtan(istart+isign,2))
           fa = (1.- (u(istart+isign,j-1,2)/vtan(istart+isign,2)))*
     +          (u(istart+isign,j-1,2)/
     +          vtan(istart+isign,2))
           delta3(istart+isign,2) = delta3(istart+isign,2) +
     +          (dy * ((fb+fa)/2.))
 701     continue
         shape(istart+isign,2) =
delta2(istart+isign,2)/delta3(istart+isign,2)
         if (tau(i,2) .le. 0.0 .and. i .ne. istop) then
c-------------------------------------------------------------------
c        this checks for the laminar separation
c-------------------------------------------------------------------
           if ((k-ksteady .ne. 0) .and. (iturb .eq. 0)) then
              print *, 'Laminar separation.  Solution returning.'
              print *, 'i = ', i
           print *, 'Time = ',t
              do m = i, istop, isign
                 delta2(m,2) = 0
              enddo
              isep = i
              return
           endif
           if (k-ksteady .eq. 0) then
              call findmax(tau,u,v,dmax,itau,iu,iv,nn,nj,nt,
     +             istat,iturb,istart,k,isign,imax)
              if (i .eq. istop) eps = eps/100.
              if (dmax .gt. eps) then
                          print *, 'Flow not steady.'
                          iturb=0
                          isep = 0
                          iseplam = 0
                          go to 75
              endif
         endif
         print *, 'Separation....'
         print *, 'i = ', i
         print *, 'Time = ',t
           do m = i, istop, isign
              delta2(m,2) = 0
           enddo
           isep = i
           return
         endif
```

```
 72     if (shape(i,2) .ge. 2.00 .and. shape(i,2) .le. 3.10
     +            .and. iturb .eq. 0
     +            .and.
     +            ((xbar(i-1).ge. 0.01 .and. isign .eq. 1)
     +            .or.
     +            (xbar(i).ge. 0.01 .and. isign .eq. -1))) then


             rey = abs(vtan(i,2)) * delta2(i,2) / nu(air)
c-------------------------------------------------------------------
c       calculate the critical Reynolds number based on the shape
c       factor.  this information is taken from p. 541 of
c       Rosenhead.  The graphical results are approximated using
c       a fourth order polynomial.
c-------------------------------------------------------------------
             a1 =    554.884547077422
             a2 =   -305.637315183644
             a3 =   -400.403765708399
             a4 =    137.244271837635
             a5 =    327.134839498025
             a6 =   -258.649002578856
             a7 =    59.5446175847718
             a8 =    3.54613361353859
             a9 =   -3.27161696528034
             a10 =   0.351442474509252
             temp = a1 + a2*((shape(i,2))**1.) + a3*((shape(i,2))**2.)
     +       + a4*((shape(i,2))**3.) + a5*((shape(i,2))**4.)
     +       + a6*((shape(i,2))**5.) + a7*((shape(i,2))**6.)
     +       + a8*((shape(i,2))**7.) + a9*((shape(i,2))**8.)
     +       + a10*((shape(i,2))**9.)
             reyc = 10.**(temp)
             test = log(rey)/log(10.)

             if (test .ge. 1.10*temp) then
             Print *, 'Transition reached at position:', x(i,1)
             iturb = i
c            if(abs(iturb-itransflag) .gt. 1 .and. k-ksteady .ne. 0
c     +           .and. isign .eq. -1) then
c                print *, 'Transition messed up.'
c                print *, 'Previous transition at:', x(itransflag,1)
c                iseplam = i
c                return
c            endif
             endif
         endif
 10      continue
c-------------------------------------------------------------------
c       a final check at istop
c-------------------------------------------------------------------
             if (k-ksteady .eq. 0 .and. i .eq. istop) then
```

```
            call findmax(tau,u,v,dmax,itau,iu,iv,nn,nj,nt,
     +        istat,iturb,istart,k,isign,imax)
              if (i .eq. istop) eps = eps/100.
              if (dmax .gt. eps) then
                   iturb = 0
                   iseplam = 0
                   isep = 0
                   go to 75
              endif
              endif
           call yfcheck(nn,nj,nt,k,delta,yf,ratio,u,v,istart,istop,y,
     +     yold,uold,vold,iflag,ksteady,isign,dydt,yoldg,deltat)
           if (iflag .eq. 1) then
                     call adjust(nn,nj,nt,k,delta,yf,ratio,
     +                    u,v,istart,istop,y,
     +                    yold,uold,vold,iflag,ksteady,isign,
     +                    yoldg,dydt,deltat)
           if (isign .eq. 1) then
                     call rk2(nn,nj,nt,y,u,v,B,istart,k,B0,acc,
     +                      isign,x,vtan)
              endif
c             if (isign .eq. -1) then
c                  call changegrid(v,y,nn,nj,nt,istart,k,li,lo,isign)
c             endif
              iturb = 0
              isep = 0
              iseplam = 0
              iflag = 0
              go to 75
           endif
           if (isep .eq. 0) then
           call output1(k+1, nn+1, nj, nt, deltat, x, tau, delta,
     +         delta2, delta3, shape,vtan,h,istart,istop,isign,u,t)
           call output2(k+1, nn+1, nj, nt, deltat, x, y, delta,u,v,
     +         vtan,tau,istart,istop,isign,t)
           endif
 999       itransflag = iturb
           return
           end
c-----------------------------------------------------------------------
c-----------------------------------------------------------------------
c-----------------------------------------------------------------------
           subroutine changegrid(v,y,nn,nj,nt,istart,k,ytemp,vtemp,isign)
           dimension ytemp(nj), vtemp(nj)
           dimension v(nn+1,nj,2), y(nn+1,nj)

           if (isign .eq. -1) then
           do 10 j = 1, nj
           ytemp(j) = y(istart,j)
           y(istart,j) = y(istart-1,j)
           vtemp(j) = v(istart,j,2)
 10        continue
           do 210 j = 2, nj
c-----------------------------------------------------------------------
c          Sweep through and find yoldp and yoldm.  Note, we don't have
```

```
c       to do the bottom values as they remain the same.
c-----------------------------------------------------------------
             do 220 m = 2,nj
                  if (ytemp(m) .ge. y(istart,j)) then
                        yoldp = ytemp(m)
                        yoldm = ytemp(m-1)
                        voldp = vtemp(m)
                        voldm = vtemp(m-1)
                        go to 230
                  endif
 220         continue
 230     slope = (voldp - voldm)/(yoldp-yoldm)
         b = ((voldm*yoldp)-(voldp*yoldm))/(yoldp-yoldm)
         v(istart,j,2) = slope*y(istart,j) + b
 210     continue

         do j = 1, nj
         y(istart,j) = ytemp(j)
         enddo

         do 11 j = 1, nj
         ytemp(j) = y(istart,j)
         y(istart,j) = y(istart+isign,j)
         vtemp(j) = v(istart,j,1)
 11      continue
         do 211 j = 2, nj
c-----------------------------------------------------------------
c       Sweep through and find yoldp and yoldm.  Note, we don't have
c       to do the bottom values as they remain the same.
c-----------------------------------------------------------------
             do 221 m = 2,nj
                  if (ytemp(m) .ge. y(istart,j)) then
                        yoldp = ytemp(m)
                        yoldm = ytemp(m-1)
                        voldp = vtemp(m)
                        voldm = vtemp(m-1)
                        go to 231
                  endif
 221         continue
 231     slope = (voldp - voldm)/(yoldp-yoldm)
         b = ((voldm*yoldp)-(voldp*yoldm))/(yoldp-yoldm)
         v(istart,j,1) = slope*y(istart,j) + b
 211     continue
         endif
         if (isign .eq. 1) then
         do 311 j = 1, nj
         ytemp(j) = y(istart-1,j)
         vtemp(j) = v(istart,j,1)
 311     continue

         mold = 0
         do 411 j = 2, nj
c-----------------------------------------------------------------
c       Sweep through and find yoldp and yoldm.  Note, we don't have
c       to do the bottom values as they remain the same.
```

```
c-----------------------------------------------------------------
                do 321 m = 2,nj
                        if (ytemp(m) .le. y(istart,j) .and.
     +                  m .gt. mold) then
                        mold = m
                                yoldp = ytemp(m)
                                yoldm = ytemp(m-1)
                                voldp = vtemp(m)
                                voldm = vtemp(m-1)
                                go to 331
                        endif
 321            continue
 331    slope = (voldp - voldm)/(yoldp-yoldm)
        b = ((voldm*yoldp)-(voldp*yoldm))/(yoldp-yoldm)
        v(istart,j,1) = slope*y(istart,j) + b
 411    continue

        endif
        return
        end

        subroutine reload(nn,nj,u,v,vtan,tau,delta,delta2,delta3,shape,
     +  h,tauw)
        dimension u(nn+1,nj,2), v(nn+1,nj,2), vtan(nn+1, 2)
        dimension tau(nn+1,2), delta(nn+1,2), delta2(nn+1,2)
        dimension delta3(nn+1,2), shape(nn+1,2), h(nn+1,2)
        dimension tauw(nn+1,2)

        do 10 i = istag+1, nn+1
        vtan(i,1) = vtan(i,2)
        vtan(i,2) = 0.
        tau(i,1) = tau(i,2)
        tau(i,2) = 0.
        tauw(i,1) = tauw(i,2)
        tauw(i,2) = 0.
        delta(i,1) = delta(i,2)
        delta(i,2) = 0.
        delta2(i,1) = delta2(i,2)
        delta2(i,2) = 0.
        delta3(i,1) = delta3(i,2)
        delta3(i,2) = 0.
        shape(i,1) = shape(i,2)
        shape(i,2) = 0.
        h(i,1) = h(i,2)
        h(i,2) = 0.
 10     continue

        do 20 i = istag+1, nn+1
        do 30 j = 1, nj
        u(i,j,1) = u(i,j,2)
        u(i,j,2) = 0.
        v(i,j,1) = v(i,j,2)
        v(i,j,2) = 0.
 30     continue
 20     continue
```

```
      do 110 i = 1, istag-1
      vtan(i,1) = vtan(i,2)
      vtan(i,2) = 0.
      tau(i,1) = tau(i,2)
      tau(i,2) = 0.
      tauw(i,1) = tauw(i,2)
      tauw(i,2) = 0.
      delta(i,1) = delta(i,2)
      delta(i,2) = 0.
      delta2(i,1) = delta2(i,2)
      delta2(i,2) = 0.
      delta3(i,1) = delta3(i,2)
      delta3(i,2) = 0.
      shape(i,1) = shape(i,2)
      shape(i,2) = 0.
      h(i,1) = h(i,2)
      h(i,2) = 0.
110   continue

      do 120 i = 1, istag-1
      do 130 j = 1, nj
      u(i,j,1) = u(i,j,2)
      u(i,j,2) = 0.
      v(i,j,1) = v(i,j,2)
      v(i,j,2) = 0.
130   continue
120   continue
      return
      end
      subroutine vnormal(vnorm, vtan, delta2, x, nn, nj, nt,
     +      normalflag,k,deltat,dvnorm,xwork,vnormwork,istag,h,
     +      vinf,t,vnormold,vnormold2,isept,isepb,sor,ksteady,
     +      iseplamb,attack)
      dimension vnorm(nn),vtan(nn+1,2),delta2(nn+1,2),x(nn+1,nj)
      dimension vnormold(nn),vnormold2(nn)
      dimension xwork(nn),vnormwork(nn),dvnorm(nn),h(nn+1,2)
c-----------------------------------------------------------------
c     store the old values of vnorm into the dvnorm vector
c-----------------------------------------------------------------
      eps = (attack/1.)*(10**(-3.))
      temp = 0.
c-----------------------------------------------------------------
c     put the old values of vnorm into dvnorm
c-----------------------------------------------------------------
      do 2 i = 1,nn
      vnormold2(i) = vnormold(i)
2     enddo
      do 3 i = 1,nn
      vnormold(i) = vnorm(i)
3     enddo
      do 5 i = 1, nn
      dvnorm(i) = vnorm(i)
5     continue

      do 6 i = 1, istag-1
```

```
        xwork(i) = x(i,1)
        if (delta2(i,2) .ne. 0) then
        vnormwork(i) = vtan(i,2)*(delta2(i,2)+h(i,2))
        else
        vnormwork(i) = 0.
        endif
 6      continue
        do 7 i = istag, nn
        xwork(i) = x(i+1,1)
        if (delta2(i+1,2) .ne. 0) then
        vnormwork(i) = vtan(i+1,2)*(delta2(i+1,2)+h(i+1,2))
        else
        vnormwork(i) = 0.
        endif
 7      continue
c-----------------------------------------------------------------
c       first we determine the normal velocities required based on the
c       current values of vtan and delta2
c
c       we have to pay special attention to the two endpoints
c-----------------------------------------------------------------
        if (delta2(1,2) .ne. 0. .and. delta2(2,2) .ne. 0) then
        vnorm(1) = (vnormwork(1) - vnormwork(2))/(xwork(1)-xwork(2))
        endif

        if (delta2(nn+1,2) .ne. 0 .and. delta2(nn,2) .ne. 0) then
        vnorm(nn) = (vnormwork(nn) - vnormwork(nn-1))/
     +       (xwork(nn)-xwork(nn-1))
        endif
c-----------------------------------------------------------------
c       now, perform a central differences on the rest of them paying
c       attention to the stagnation point
c-----------------------------------------------------------------
        do 10 i = 2, istag-1
        dx = xwork(i)-xwork(i-1)
        alpha = (xwork(i+1) - xwork(i))/dx
        if(vnormwork(i-1) .ne. 0. .and.
     +       vnormwork(i) .ne. 0. .and.
     +       vnormwork(i+1) .ne. 0.) then
        vnorm(i) = (vnormwork(i+1) + (((alpha**2.) - 1.) *
     +       vnormwork(i)) - ((alpha**2.)*
     +       vnormwork(i-1)))/
     +       (alpha*(alpha+1.)*dx)
        endif
 10     continue
        do 15 i = istag, nn-1
        dx = xwork(i)-xwork(i-1)
        alpha = (xwork(i+1) - xwork(i))/dx
        if(vnormwork(i-1) .ne. 0. .and.
     +       vnormwork(i) .ne. 0. .and.
     +       vnormwork(i+1) .ne. 0.) then
        vnorm(i) = (vnormwork(i+1) + (((alpha**2.) - 1.) *
     +       vnormwork(i)) - ((alpha**2.)*
     +       vnormwork(i-1)))/
     +       (alpha*(alpha+1.)*dx)
```

```
        endif
 15     continue
        do 8 i = 1, nn
        dvnorm(i) = (vnorm(i) - dvnorm(i))
 8      continue

        dmax = 0.
        do i = 1,nn
        if (abs(dvnorm(i)) .ge. dmax)then
        dmax = abs(dvnorm(i))
        endif
 46     enddo
        dmax = dmax/vinf

        if(dmax .gt. 0.30 .and. k-ksteady .gt. 1) then
        sor = sor*0.90
        print *, 'Relaxation factor changed to: ', sor
        endif

c       do i = istag-1,1, -1
c            if (dvnorm(i) .eq. 0.) then
c            vnorm(i) = 1.05*vnorm(i+1)
c            endif
c       enddo
c       do i = istag, nn
c            if (dvnorm(i) .eq. 0.) then
c            vnorm(i) = 1.05*vnorm(i-1)
c            endif
c       enddo

        if (dmax .ge. eps) then
        normalflag = 1
        print *, 'Maximum difference in normal velocities: ',dmax
        print *, ' '
        endif
c-----------------------------------------------------------------------
c       write some data to file
c-----------------------------------------------------------------------
        open (unit = 7, file = 'normvel.dat', status = 'unknown')
        write(7,659) t
        write(7,*) 'Vinf:',vinf
        write(7,*) ' '
        write(7,*) 'Stagnation at station: ',istag
        write(7,700)
        do 510 i = 1, nn
        if (abs(vnorm(i)) .le. 1.*(10.**(-5.))) then
        temp = 999.
        go to 45
        endif
 45     write (7,600) i, vnormold2(i), vnormold(i),vnorm(i),dvnorm(i)
 510    continue
        close(unit = 7)
 600    format(1x,i5,4(3x,f12.7))
 659    format(1x,'Time:','(','seconds',
    +           ')',2x,f12.4)
```

```
 700     format(1x,3x,'Element',5x,'vnormold2',5x,'vnormold',5x,'vnorm',
     +        5x,'dvnorm')
c-----------------------------------------------------------------------
         return
         end

         subroutine normok(vnormok,vnorm,nn)
         dimension vnormok(nn),vnorm(nn)
         do i = 1, nn
              vnormok(i) = vnorm(i)
         enddo
         return
         end
         subroutine normswitch(vnormok,vnorm,nn)
         dimension vnormok(nn),vnorm(nn)
         do i = 1, nn
              vnorm(i)=vnormok(i)
         enddo
         return
         end
         subroutine yfcheck(nn,nj,nt,k,delta,yf,ratio,u,v,istart,istop,y,
     +   yold,uold,vold,iflag,ksteady,isign,dydt,yoldg,deltat)
         dimension yold(nn+1,nj), uold(nn+1,nj), vold(nn+1,nj)
         dimension y(nn+1,nj), u(nn+1,nj,2), v(nn+1,nj,2)
         dimension delta(nn+1,2),yoldg(nn+1,nj),dydt(nn+1,nt)

         imax = 0
         deltamax = 0.
         do 100 i = istart, istop, isign
         if (delta(i,2)+y(i,1) .gt. deltamax) then
              deltamax = delta(i,2)+y(i,1)
              imax = i
         endif
 100     continue
         if (k-ksteady .eq. 0) then
         go to 2000
         endif
         if (deltamax .eq. 0) then
         go to 2000
         endif
         print *, 'Maximum delta = ', deltamax
         print *, 'at station:', imax

         If (yf/deltamax .gt. 2.0) then
              fract = (RAN(0)-0.5)*0.2
              yf = (1.40+fract) * deltamax
              print *, 'Adjusting the grid.'
              print *, 'yf to deltamax = ', yf/deltamax
              print *, 'New yf = ', yf
              print *, ' '
              iflag = 1

              do 10 i = istart, istop, isign
              do 20 j = 1, nj
              yold(i,j) = y(i,j)
```

```
              uold(i,j) = u(i,j,1)
              vold(i,j) = v(i,j,1)
 20           continue
 10           continue


c-------------------------------------------------------------------
c        for each x station, we need to find the distance between h(x) and
c        yf to set the grid
c-------------------------------------------------------------------
              do 60 i    = istart, istop, isign
                   sum = 0.0
                   do 110 j = 0, nj-2
                   sum = sum + ratio**(j)
 110               continue
c                  dy = (yf-yold(i,1))/sum
                   dy = (yf)/sum

                   do 50 j     = 2, nj
                   y(i,j) = y(i,j-1) + ratio**(j-2) * dy
 50                continue
 60           continue
c-------------------------------------------------------------------
c        Now, we need to find the values for u and v at the new y values
c-------------------------------------------------------------------
              do 200 i = istart, istop, isign
              do 210 j = 2, nj
c-------------------------------------------------------------------
c        Sweep through and find yoldp and yoldm.  Note, we don't have
c        to do the bottom values as they remain the same.
c-------------------------------------------------------------------
              do 220 m = 2,nj
              if (yold(i,m) .gt. y(i,j)) then
                   yoldp = yold(i,m)
                   yoldm = yold(i,m-1)
                   uoldp = uold(i,m)
                   uoldm = uold(i,m-1)
                   voldp = vold(i,m)
                   voldm = vold(i,m-1)
                   go to 230
              endif
 220          continue
 230          slope = (uoldp - uoldm)/(yoldp-yoldm)
              b = ((uoldm*yoldp)-(uoldp*yoldm))/(yoldp-yoldm)
              u(i,j,1) = slope*y(i,j) + b
              slope = (voldp - voldm)/(yoldp-yoldm)
              b = ((voldm*yoldp)-(voldp*yoldm))/(yoldp-yoldm)
              v(i,j,1) = slope*y(i,j) + b
 210          continue
 200          continue
         endif
 2000    do i = 1,nn+1
         do j = 1, nj
         dydt(i,j) = 0.
         yoldg(i,j) = y(i,j)
```

```
      enddo
      enddo
      return
      end

      subroutine adjust(nn,nj,nt,k,delta,yf,ratio,u,v,istart,istop,y,
     +  yold,uold,vold,iflag,ksteady,isign,yoldg,dydt,deltat)
      dimension yold(nn+1,nj), uold(nn+1,nj), vold(nn+1,nj)
      dimension y(nn+1,nj), u(nn+1,nj,2), v(nn+1,nj,2)
      dimension delta(nn+1,2),yoldg(nn+1,nj)
      dimension dydt(nn+1,nj)
           yf = 1.25* yf
           print *, 'Adjusting the grid.'
           print *, 'New yf = ', yf
           print *, ' '
           iflag = 1

           do 10 i = istart, istop, isign
           do 20 j = 1, nj
           yold(i,j) = y(i,j)
           uold(i,j) = u(i,j,1)
           vold(i,j) = v(i,j,1)
 20        continue
 10        continue


c-----------------------------------------------------------------
c       for each x station, we need to find the distance between h(x) and
c       yf to set the grid
c-----------------------------------------------------------------
           do 60 i    = istart, istop, isign
                sum = 0.0
                do 110 j = 0, nj-2
                sum = sum + ratio**(j)
 110            continue
c               dy = (yf-yold(i,1))/sum
                dy = (yf)/sum

                do 50 j     = 2, nj
                y(i,j) = y(i,j-1) + ratio**(j-2) * dy
 50             continue
 60        continue
c-----------------------------------------------------------------
c       Now, we need to find the values for u and v at the new y values
c-----------------------------------------------------------------
           do 200 i = istart, istop, isign
           do 210 j = 2, nj
c-----------------------------------------------------------------
c       Sweep through and find yoldp and yoldm.  Note, we don't have
c       to do the bottom values as they remain the same.
c-----------------------------------------------------------------
           do 220 m = 2,nj
           if (yold(i,m) .gt. y(i,j)) then
                yoldp = yold(i,m)
                yoldm = yold(i,m-1)
```

```
                uoldp = uold(i,m)
                uoldm = uold(i,m-1)
                voldp = vold(i,m)
                voldm = vold(i,m-1)
                go to 230
            endif
220         continue
230         slope = (uoldp - uoldm)/(yoldp-yoldm)
            b = ((uoldm*yoldp)-(uoldp*yoldm))/(yoldp-yoldm)
            u(i,j,1) = slope*y(i,j) + b
            slope = (voldp - voldm)/(yoldp-yoldm)
            b = ((voldm*yoldp)-(voldp*yoldm))/(yoldp-yoldm)
            v(i,j,1) = slope*y(i,j) + b
210         continue
200         continue
2000    do i = 1, nn+1
        do j = 1, nj
        dydt(i,j) = 0.
        yoldg(i,j) = y(i,j)
        enddo
        enddo

        return

        end
c-------------------------------------------------------------------------
c       Set the pointers
c       ----------------
c       From the potential flow problem
c       -------------------------------
c       x(nn+1)                 :i1
c       y(nn+1)                 :i2
c       xbar(nn)                :i3
c       ybar(nn)                :i4
c       rlen(nn)                :i5
c       theta(nn)               :i6
c       r(nn,nn+1)              :i7
c       a(nn+1,nn+1)            :i8
c       b(nn+1)                 :i9
c       ipvt(nn+1)              :i10
c       work(nn+1)              :i11
c       vtan(nn)                :i12
c       cp(nn)                  :i13
c       beta(nn,nn)             :i14
c       q(nn+1)                 :i15
c       phi(nn)                 :i16
c       phis(nn)                :i17
c       phiv(nn)                :i18
c       phiold(nn)              :i19
c       bigwork(nn+1,nn+1)      :i20
c       circ(nt)                :i21
c       circx(nt,nt)            :i22
c       circy(nt,nt)            :i23
c-------------------------------------------------------------------------
c       From the gas-dynamic boundary layer module
```

```
c       ------------------------------------------
c       x((nn+1),nj)            :i24
c       y((nn+1),nj)            :i25
c       u((nn+1),nj,2)          :i26
c       v((nn+1),nj,2)          :i27
c       vtan((nn+1),2)          :i28
c       aa(nj)                  :i29
c       bb(nj)                  :i30
c       cc(nj)                  :i31
c       dd(nj)                  :i32
c       tau((nn+1),2)           :i33
c       delta((nn+1),2)         :i34
c       delta2((nn+1),2)        :i35
c       delta3((nn+1),2)        :i36
c       shape((nn+1),2)         :i37
c       li(nj)                  :i38
c       lo(nj)                  :i39
c       lm(nj)                  :i40
c       yplus(nj)               :i41
c
c
c       From the deicing fluid module
c       -----------------------------
c       h((nn+1),2)             :i42
c       f(nn+1)                 :i43
c       aaf(nn+1)               :i44
c       bbf(nn+1)               :i45
c       ddf(nn+1)               :i46
c       work1(4)                :i47
c       work2(4)                :i48
c       work3(3)                :i49
c
c
c       From the grid module
c       --------------------
c       f((nn+1),nj)            :i50
c       crap((nn+1),nj)         :i51
c       crap((nn+1),nj)         :i52
c       crap((nn+1),nj)         :i53
c       crap((nn+1),nj)         :i54
c       crap((nn+1),nj)         :i55
c       crap((nn+1),nj)         :i56
c       xoldg((nn+1),nj)        :i57
c       yoldg((nn+1),nj)        :i58
c       dxdt((nn+1),nj)         :i59
c       dydt((nn+1),nj)         :i60
c       crap((nn+1),nj)         :i61
c       crap((nn+1),nj)         :i62
c       crap((nn+1),nj)         :i63
c       crap((nn+1),nj)         :i64
c       crap((nn+1),nj)         :i65
c       crap((nn+1),nj)         :i66
c
c       MISC.
c       -----
```

```
c       dtau((nn+1))          :i67
c       dus((nn+1))           :i68
c       yold(nn+1,nj)         :i69
c       uold(nn+1,nj)         :i70
c       vold(nn+1,nj)         :i71
c       vnorm(nn)             :i72
c       dvnorm(nn)            :i73
c       xwork(nn)             :i74
c       vnormwork(nn)         :i75
c       tauw(nn+1,2)          :i76
c       vnormold(nn)          :i77
c       vnormold2(nn)         :i78
c       htemp(nn+1,2)         :i79
c       vnormok(nn)           :i80
c
c-------------------------------------------------------------------------
c       files used in final code
c
c       coef.f
c       control.f
c       cpv.f
c       createinput.f
c       curve.f
c       cylinder.f
c       decomp.f
c       derivs.f
c       dfluid.f
c       driver.f
c       findh.f
c       findstag.f
c       findstart.f
c       functions.f
c       geominput.f
c       getnn.f
c       grid.f
c       initialgrid.f
c       liftdrag.f
c       ode.f
c       output.f
c       potentials.f
c       potoutput.f
c       reader.f
c       rk.f
c       setup.f
c       soln.f
c       solvit.f
c       tanvel.f
c       thomas.f
c       transfervtan.f
c       usolve.f
c       vnormal.f
c       yfcheck.f
c       makefile
c-------------------------------------------------------------------------
```